Research Paper

A Competitive Nature-Inspired Cuckoo Search Algorithm for Solving the Resource Constrained Project Scheduling Problem

Seyed Taha Hossein Mortaji*, Mohammadreza Hosseinzadeh*

* PhD., Iran University of Science and Technology, Industrial Engineering Department, Tehran, Iran.

**ARTICLE INFO**

**Received:** 24 July 2021
**Reviewed:** 7 August 2021
**Revised:** 31 August 2021
**Accepted:** 7 September 2021

**KEYWORDS:**
Resource Constrained Project Scheduling, Meta-heuristics, Cuckoo Search, Levy Flights, Makespan Minimization.

**ABSTRACT**

Since its advent, the resource constrained project scheduling problem has been a favorite NP-Hard problem studied by many researchers. In this paper, we examine this problem with a focus on the total project time minimization, usually known as the makespan minimization. To this end, a nature-inspired Cuckoo Search Algorithm (CSA) has been introduced that works by simulating the brooding parasitism of some cuckoo birds in nature. In a nutshell, the proposed cuckoo search algorithm has two main capabilities including: a) a global random search using Levy flights, and b) a local random refinement strategy. In addition, a double justification method has been employed to improve the generated solutions. Simultaneous use of these features can maximize the efficiency of resource searches in uncertain environments. Experimental results on the standard J30, J60, and J120 test sets from the PSPLIB are also presented, showing that the CSA has satisfying results compared with the other well-reported algorithms.

1. Introduction

The critical path calculation of a project network ignores the fact that renewable resources are limited in availability. When the required amount of these resources are exceeded than their availability, over-allocation or/and conflicts may occur at some time instances in the project execution. In this case, some activities need to be delayed to construct a feasible schedule respecting the precedence relations and the limited availability of renewable resources. The presence of such constraints and the act of constructing a precedence- and resource-feasible schedule often leads to a complex scheduling problem which is usually known as the Resource-Constrained Project Scheduling Problem (RCPSP) (Humyun Fuad, et al., 2020).

*Corresponding Author
Mortaji@ymail.com

48
Despite being hard-to-solve, RCPSP may be readily formulated as follows. A project consists of \( n \) activities \( i=1,\ldots,n \) and \( r \) renewable resources. Each activity \( i \) has to be processed to complete the project by taking \( t_i \) units of time. Activity \( i \) requires \( r_{ik} \) units of resource \( k \) to be completed. However, resources have a limited periodic availability such that only a constant amount of \( R_k \) units of resource \( k \) is available at any point in time. There is a set of predecessors for each activity \( i \) represented by the set \( B \). \( B \) implies that activity \( i \) is forced to be started only once all of its predecessors in \( B \) have been completed. Therefore, the activities are impacted by two kinds of constraints. First, the precedence constraints, which force each activity \( i \) to be scheduled after all of its predecessor activities. Second, performing the activities requires resources with limited availabilities.

During the execution phase, splitting an activity (known as activity preemption) is not allowed which means that, each activity, once started, should continue until the end of its duration. Moreover, the standard RCPSP assumes that each activity has a fixed duration mode meaning that the estimated duration and resource demands of activities cannot be changed during the project (Vanhoucke, 2012; Brucker & Knust, 2012; Nozari et al., 2016; Shali et al., 2021).

The RCPSP can be represented as an activity-on-node (AoN) network. In this representation, the activities 0 and \( n+1 \) are dummy, have no duration, require no resources, and denote the start and end of the project, respectively. As mentioned earlier, a schedule is said to be feasible only if the precedence and the renewable resource constraints are met (Zaman, et al., 2021; Shirzi 2021). So, as a regular measure of performance, the objective of the RCPSP is to find a feasible schedule, say \( S \), by minimizing the start time of the dummy end activity \( n+1 \). Fig. 1 gives an example of a project with \( n=9 \) non-dummy activities and finish to start precedence relations between them on an AoN network. This project has to be scheduled respecting one renewable resource with a capacity of 2 units. Each node represents an activity, and the numbers above and below the node denote the estimated activity duration and the resource demands, respectively.

![Fig. 1. A Project Example](image-url)

Despite being hard to make a precedence- and resource-feasible schedule, the RCPSP is a very applicable scheduling problem that has attracted increasing interest in both academic and engineering fields (Pellerin, et al., 2020; Ben Issa & Tu, 2020; Nozari & Szmelter 2018). Many complicated scheduling problems such as job shop scheduling, flow-shop scheduling, and assembly line balancing may be subsumed from the classical RCPSP (Artigues, et al., 2011; Brucker, 2002; Jamshidi and Sadeghi 2021). It also may be used to model many practical situations like scheduling a production process, a software project, a school timetable, the construction of a house, or the renovation of an airport (Menesi & Hegazy, 2014; Chiong & Siarry, 2012; Aliahmadi et al., 2015). Over the last few decades, there has been an increasing interest for solving the RCPSP, and it has been considerably developed (Gan & Xu, 2013; Bettemir & Sonmez, 2014; Vanhoucke, 2012; Brucker & Knust, 2012; Nozari et al., 2016; Shali et al., 2021).
Kolisch & Padman, 2001). Also, a significant number of exact algorithms to solve this scheduling problem respecting the precedence relations and the periodic availability of resources are proposed in the literature (Coelho & Vanhoucke, 2020). Currently, among the exact algorithms to solve the RCPSP, the most competitive algorithms are Zamani (2001), Aristide et al. (1998), and Bianco and Caramia (2012).

It has been shown by Blazewicz et al. (1983) that the RCPSP belongs to the class of NP-hard optimization problems in a strong sense. Hence, even the most efficient exact algorithms such as branch and bound, and constraint-propagation-based cutting planes can only solve small-sized problems with less than 60 activities within adequate computation time (Demassey, et al., 2005; Brucker & Knust, 2003; Watermeyer & Zimmermann, 2020). Therefore, the use of meta-heuristic solution procedures for solving large problem instances is justified to obtain optimal or near-optimal solutions within an acceptable amount of time (Diana & De La Garza, 2020). Excellent introductions and extensive surveys on the different heuristics for solving the RCPSP can be found in Hartmann and Briskorn (2010).

This paper introduces an effective Cuckoo Search Algorithm (CSA) for solving the RCPSP with a focus on the makespan minimization. Our implemented algorithm cleverly uses a local random walk while employing the global explorative random walk using Levy Flights. Simultaneous use of the Local and general random walks leads to a more efficient exploration of the search space and as a result, a higher probability of finding the global optimum solution will be achieved. Moreover, once a feasible schedule is constructed, a double justification method improves the initial solutions (Mohammadi et al., 2015).

The rest of this paper proceeds as follows. In section 2, the solution representation (encoding), and the procedure used to construct a feasible schedule (decoding) are briefly explained. Section 3 has been devoted to describing the implemented structure of the proposed CSA for the RCPSP. Section 4 begins by reporting the results of the computational experiments and looks at how much the cuckoo search algorithm is capable of finding minimal makespans compared with the best-reported algorithms. The fifth section is concerned with finding the appropriate parameter values by conducting a general full factorial design in which makespan is measured at all combinations of the experimental factor levels. Finally, the sixth section presents a conclusion on the findings of the research and takes a look at the future research avenues.

### 2. Solution Representation (Encoding) and Schedule Generation Scheme (Decoding)

Most of the heuristics used for solving the resource-constrained project scheduling problems are categorized into two classes including a) priority rule-based methods, and b) meta-heuristic-based approaches (Kolisch & Hartmann, 1999). The first class, also known as constructive methods, starts with one of the activities being scheduled. Subsequently, in each step, a single schedule is constructed until all activities have been considered. Several approaches of this class have been suggested and are available in the literature, e.g., Alvarez-Valdes and Tamarit (1989), Boctor (1990), Kolisch (1996), and Tormos and Lova (2001). In the second class, usually known as improvement methods, an initial solution is improved by successively performing operations in which one or several solutions are transformed into others. Several approaches of this class have also been proposed in the literature, e.g., Hartmann (2002), Kochetov and Stolyar (2003), and Valls et al. (2008).

The spine of most of the methods for solving the RCPSP, where an initial group of solutions is gradually improved, is the schedule representation and generation schemes. Typically, it is more convenient and practical to operate on the particular representation of a solution (encoded solution) rather than directly work on the constructed solution. For this purpose, five solution representation schemes including Activity List Representation, Random Key Representation, Priority Rule Representation, Schedule Scheme Representation, and Shift Vector Representation have been described by Kolisch and Hartmann (1999). Each of these representation schemes assigns priorities to project activities and generates a precedence-feasible representation of the solution by creating a ranking. However, among these schemes, the two most
promising ones seem to be the random-key (RK) and the activity list (AL) representations (Kolisch & Hartmann, 2006). In the RK representation, a solution corresponds to a point in the Euclidian space, such that the $i^{th}$ element corresponds to the priority value of the $i^{th}$ activity. In the AL representation, each activity is scheduled after all of its predecessors specified in the set B. In this way, a precedence-feasible activity list is constructed. For a particular schedule, the RK and the AL representations may be different. Therefore, neither the AL nor the RK guarantees to provide an ideal representation. In this paper, we opt to use the RK representation in the proposed CSA.

The schedule generation scheme determines how the activities’ start times are assigned to the previously constructed priority lists and as a result, a feasible solution is constructed. There are two alternatives for the schedule generation schemes. In a serial schedule generation scheme (SSGS) activity-incrementation approach is followed and activities are selected one by one from the list and are scheduled as soon as possible, while in a parallel schedule generation scheme (PSGS) time-incrementation approach is used and the activities are scheduled on the list as long as resource constraints are satisfied. Kolisch (1996) showed that, although the PSGS sometimes works better, the serial method is superior for large sample sizes and for instances that are only moderately resource-constrained. However, to take advantage of both schemes, this paper uses a stochastic scheduling generation scheme that works between the serial and parallel SGSs. The next section represents an overview of the proposed CSA and shows how it works in practice.

3. Cuckoo Search Algorithm for the RCPSP

The cuckoo search algorithm was firstly introduced by inspiring from the offspring parasitism of cuckoo birds in which the mother replaces her egg with the host egg (Yang & Deb, 2009; Yang & Deb, 2010). In addition to the simple isotropic random walk, the cuckoo search algorithm has been enhanced by the use of the Levy flights which makes it more compelling. Owing to its capability in solving different kinds of optimization problems, the cuckoo search algorithm has attracted a lot of attention from researchers and has been found in many real-world applications.

Nowadays, cuckoo search is applied by researchers in many academic areas such as engineering optimization, image processing, scheduling, planning, feature selection, and forecasting (Yang & Deb, 2010). It also has been employed for solving many real-world applications such as wind turbine blades (Ernst, et al., 2012), reliability problems (Valian, et al., 2013), wireless sensor networks (Dhivya & Sundarambal, 2011), flow shop scheduling (Marichelvam, 2012), traveling salesman problem (Ouaarab, et al., 2013), and job scheduling in the grid (Prakash, et al., 2012). Since its advent, many variants of the cuckoo search algorithm have been developed among them the major variants appear to be the quantum-inspired CS (Layeb, 2011), modified CS (Walton, et al., 2011), multi-objective CS (Yang & Deb, 2011), discrete CS (Jati & Manurung, 2012), modified adaptive CS (Zhang, et al., 2012), and neural-based CS (Khan & Sahai, 2013).

The CSA uses the following three simplified rules: 1) each cuckoo lays one egg at a time and conveys it in a randomly chosen habitat, 2) the best habitats with the highest quality eggs will be transmitted over the generations, and 3) the number of available host habitats is assumed to be fixed. The host bird may discover the egg laid by a cuckoo with a probability of $p_a \in [0,1]$. In this case, the host bird can either throw out the egg or simply desert the nest and build a completely new nest. The above-mentioned rules can be expressed in mathematical terms as follows. Each egg in a habitat represents a priority vector $x \in R^n$, (known as the random key vector), corresponds to a solution; and each solution corresponds to a point in the n-dimensional Euclidean space. If $l_i$ and $u_i$ are the lower and upper bounds of activity $i$, priority elements of an RK vector can have real values between 1 and $n$, where $n$ is the total number of activities, $l_i = 0$ and $u_i = n$. A higher value in the RK vector implies a lower priority to get into the schedule.
Because each activity must necessarily be scheduled after its predecessors, we have decreased the solution space by replacing the previous $l_i$s and $u_i$s with new lower and upper bounds. For this purpose, $P_i$ and $S_i$ are considered to be the number of predecessors and successors of activity $i$, respectively. The new lower and upper bounds for activities’ priorities can now be calculated through $l_i = P_i + 1$, and $u_i = n - S_i$. In this way, a solution with a precedence-feasible priority structure is obtained, where each activity has a higher RK value or a lower priority, than all of its predecessors. The reduced solution space enables the CSA to perform its mathematical operations more efficiently.

The priority vectors move in the Euclidian space based on a Levy or random walk, and feasible schedules are constructed. According to the second rule, not-so-good solutions are identified over the generations and replaced by new and potentially better solutions. The following subsections explain how the CSA moves toward the optimal or near-optimal solution. The pseudo-code of the CSA has also been presented in Figure 3.

### 3.1. Random initial solutions

In this step, an initial population $X = \{x_1^0, x_2^0, ..., x_N^0\}$ including $N$ cuckoos is randomly generated where vector $x_i^0 \in \mathbb{R}^n$ is the priority vector and corresponds to a solution. Concerning the upper and lower values, precedence-feasible priority values are obtained for each activity $i$. Then, the generated values are evaluated to find the current global best. In this way, a feasible schedule without violating both the precedence and resources constraints is constructed by assigning start times to the project activities, and concerning the schedule generation schemes, i.e., serial and parallel schedule generation schemes. However, deviating from the general belief, Kolisch (1996) showed that the parallel method does not generally perform better than the serial method. Rather, it gives only better results for small sample sizes as well as for "hard" (that is highly resource-constrained) problems. Hence, the serial method is superior for large sample sizes and for instances that are only moderately resource-constrained, and in the other cases, the parallel method works better. Hence, this paper uses a scheduling generation scheme that works between the serial and parallel schedule generation schemes and is controlled by a switching parameter or simply with a random number $r \in [0,1]$. Contrary to the two ordinary generation schemes, by using this mechanism, the schedules can be generated between non-delay and active ones and therefore can adjust their behavior for various problem instances.

### 3.2. Global explorative random walk using Levy flights

In this step, the primary mobility of the CSA is performed by exploration through the global search mechanism of the Levy flights. Studies show that Levy flights can maximize the efficiency of resource searches in uncertain environments. Broadly speaking, Levy flights are a random walk whose step length is drawn from the Levy distribution. A simple version of the Levy distribution can be defined as:

\[
\begin{align*}
\mathcal{L}(\mu, s, \gamma) &= \sqrt{\frac{\gamma}{2\pi s \Gamma(s)}} \int_0^{\infty} e^{-\frac{\gamma (u - \mu)^2}{2s \Gamma(s)}} du \\
&= 0 \quad \text{otherwise}
\end{align*}
\]

where $\mu > 0$ indicates a minimum step, $s$ indicates the step size, and $\gamma$ indicates a scale parameter. From a practical perspective, random numbers with Levy flights are generated in two stages. The first stage is the choice of a random direction and the second is the generation of steps that follows the chosen Levy distribution. The generation of a direction $d$ can be easily drawn from a uniform distribution, while the generation of steps is quite tricky. There are a few ways of generating steps, and one of the most efficient and yet straightforward ways appears to be the Mantegna algorithm for a symmetric (means that the steps...
can be positive and negative) Levy stable distribution (Mantegna, 1994). In Mantegna’s algorithm, the step $s$ is calculated as follows:

$$ s = \frac{u}{|v|^\beta} $$  

(2)

In which, $u$ and $v$ are generated from the normal distribution, thus:

$$ u \sim N(0, \sigma_u^2) $$

(3)

and

$$ v \sim N(0, \sigma_v^2) $$

(4)

where,

$$ \sigma_u = \left( \frac{\Gamma(1+\beta)\sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right)\beta^{2(\beta-1)}(\beta-1)^2} \right)^{\frac{1}{\beta}} $$

(5)

and

$$ \sigma_v = 1 $$

(6)

Once the step is calculated, each of the solutions (cuckoos or habitats) is updated according to the current best solution with minimum makespan. Also, to make some noise in the previous solution, a random number $\varepsilon$ is employed which is drawn from a Uniform distribution (see equation 7). The difference factor in Equation 7, i.e. $(x_i^t - Best)$, means that when the solution is the best it remains unchanged. Then, the updated habitats are evaluated to update the current global best.

$$ x_i^{t+1} = x_i^t + \varepsilon \times s \times d \times (x_i^t - Best) $$

(7)

### 3.3. Local random refinement

To do a local refinement, in this step, a fraction of worse habitats is discovered with a probability $p_a$, and are updated using the local random walk mechanism. In the real world, if a cuckoo’s egg is very similar to a host’s eggs, then this cuckoo’s egg is less likely to be discovered, thus it is recommended that the fitness should be related to the difference in solutions. Therefore, it is a good idea to do a random walk in a biased way with some random step sizes. For the implementation point of view, a random number $r$ can be drawn from a Uniform distribution and if $r < p_a$, the local random walk can be performed as follows (it is clear that if $r > p_a$, $x_i^{t+2} = x_i^{t+1}$):

$$ x_i^{t+2} = x_i^{t+1} + \varepsilon \times (x_j^{t+1} - x_k^{t+1}) $$

(8)

where $x_j^{t+1}$ and $x_k^{t+1}$ are two different solutions selected randomly by random permutation, and $\varepsilon$ is a random number drawn from Uniform distribution. Once the random walk is carried out, the best solutions are evaluated and then transmitted to the next step for improvement.

### 3.4. Improvement

In this step, CSA uses an improvement method to transform a solution into one or more enhanced solutions. The iterative forward/backward method is known as the most efficient mechanism for improving solutions of the RCPSP (Li & Willis, 1992). There are a few major variants of this approach, among them, the double justification method has proved to be very efficient (Valls, et al., 2005). Justification is a simple technique that can be incorporated easily into various algorithms. It produces notable improvements in the
quality of the schedules with a small and favorable change in computing time. Double justification has two main steps, namely, forward improvement and backward improvement. A forward improvement produces a right-justified schedule through backward-scheduling of activities according to the non-increasing order of their finish times. In other words, in the forward pass, the activities are sorted according to the non-increasing order of their finish times. Afterward, the activities are considered from right to left, and they are scheduled at the latest feasible time. After the forward pass, the produced schedule is converted to a left-justified one in the backward pass. That is, the activities are sorted in non-decreasing order of their start times, and they take place from the project’s start time (time zero in the project horizon) at the earliest feasible time.

In Fig. 2, a single iteration step is used to illustrate the procedure of the forward-backward improvement. To reduce the makespan of the given schedule, consider a feasible schedule of the project example in Fig. 1, with a makespan of 18 units (see Fig. 2.a). At first, the forward improvement shifts the activities to right, as much as possible, and on the order of their finish times. In this way, a schedule with a makespan of 16 units is obtained (see Fig. 2.b). Afterward, the backward improvement shifts the activities to left, as much as possible, and on their start times. Moreover, finally, an improved schedule with a makespan of 15 units is obtained (see Fig. 2.c).

![Fig. 2. (a) A feasible schedule of the project example using a serial generation scheme, (b) Right justified (forward) schedule, (c) Left justified (backward) schedule](image-url)
The procedure of the proposed CSA can be summarized as the pseudo-code shown in Fig. 3.

---

**Objective:** makespan minimization

**Pre-process and initialization**

*Generate initial solution considering lower and upper bounds*

*Find the current best solution with respect to the generation scheme*

While \((t < \text{MaxGeneration}) \) or \((\text{stopping criteria})\)

Global explorative random walk using Levy flights

*Move towards the best solution with a random direction and a stochastic step size*

*Update the current best solution with respect to the proposed generation scheme*

Local random refinement

*Abandon a fraction \((Pa)\) of worse solutions and generate new solutions*

*Update the current best solution with respect to the proposed generation scheme*

*Improve the solution using double justification improvement method*

*Create a right justified schedule using backward scheduling*

*Create a left justified schedule using forward scheduling*

*Update the current best solution with respect to the generation scheme*

End while

**Post-process results and visualization**

---

Fig. 3. The pseudo-code of the cuckoo search algorithm.

### 4. Choice of Parameters

Like other meta-heuristics, CSA has a predefined set of parameters that has to be initialized before execution, concerning the problem at hand. An appropriate initial parameter setting, usually, has a significant impact on the solving progress, such as the diversification or intensification rate of the search space, and therefore the quality of the solution and the time required to achieve it. The importance of parameter tuning is discussed in Eiben (1992). To the best of our knowledge, for most of the meta-heuristics, there exists no general optimal initial parameter setting. Hence, an optimal initial parameter setting can vary considerably from problem to problem, and even between problem instances. Various studies and applications have implied that cuckoo search has a robust behavior regarding parameters; that is, the convergence rate, to some extent, is not sensitive to the parameters used (Wang, et al., 2012). For the sake of clarity, we have examined the parameters by carrying several experiments out on test instances of the PSPLIB, consisting of the groups of 45, 41, and 51 from the sets J60, J90, and J120, respectively. Each of these groups includes ten benchmark instances. It is worth mentioning that based on the total gap existing between the latest lower and upper bounds of their ten instances, in the PSPLIB, updated in the year 2010, each of these three groups can be considered as the hardest group in its own benchmark set. This approach for parameter tuning has been used in Zamani (2001).

CSA has four parameters to be tuned that are: the number of habitats, i.e., the population size \(N\), the fraction of worst habitats to be discovered, \(Pa\), the step size scaling factor \(\alpha\), and \(\beta\) used in the Magneta’s algorithm. In this section, to find appropriate parameter values, a general full factorial design has been conducted in which makespan is measured at all combinations of the experimental factor levels. For this
purpose, the population size is changed from 10 to 40, the probability Pa is changed from 0.25 to 0.75, the step size scaling factor, α, is changed from 0.1 to 1, and β is changed from 1 to 4. The combinations of factor levels represent the conditions at which responses will be measured. Since the experimental runs include all combinations of the factor levels, a general full factorial design is generally used in optimization experiments.

For example, if we limit the number of schedules to 1000, Table 1 is constructed that shows the analysis of variance (ANOVA) table of group 51 of the set 120. The p-value of 0.000 for β and α indicates that these factors are significant compared with an alpha value of 0.01. However, p-values of 0.118, 0.770, and 0.897 for 2-way, 3-way, and 4-way interactions indicate that they are not significant. Therefore, it concludes that these interactions do not affect the response (makespan). It can be seen in the interaction plot for makespan which has been presented in Fig. 4. Parallel lines in this figure indicate no significant interaction between the factors.

Table 1. Analysis of variance (ANOVA) table for algorithm parameters

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Adj SS</th>
<th>Adj MS</th>
<th>F-Value</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>359</td>
<td>1389.42</td>
<td>3.870</td>
<td>1.08</td>
<td>0.167</td>
</tr>
<tr>
<td>Linear</td>
<td>14</td>
<td>222.13</td>
<td>15.867</td>
<td>4.43</td>
<td>0.000</td>
</tr>
<tr>
<td>N</td>
<td>3</td>
<td>8.63</td>
<td>2.876</td>
<td>0.80</td>
<td>0.492</td>
</tr>
<tr>
<td>Beta</td>
<td>5</td>
<td>85.67</td>
<td>17.135</td>
<td>4.79</td>
<td>0.000</td>
</tr>
<tr>
<td>Pa</td>
<td>2</td>
<td>28.00</td>
<td>14.000</td>
<td>3.91</td>
<td>0.020</td>
</tr>
<tr>
<td>Alpha</td>
<td>4</td>
<td>99.83</td>
<td>24.958</td>
<td>6.98</td>
<td>0.000</td>
</tr>
<tr>
<td>2-Way Interactions</td>
<td>71</td>
<td>306.96</td>
<td>4.323</td>
<td>1.21</td>
<td>0.118</td>
</tr>
<tr>
<td>N*Beta</td>
<td>15</td>
<td>98.32</td>
<td>6.555</td>
<td>1.83</td>
<td>0.026</td>
</tr>
<tr>
<td>N*Pa</td>
<td>6</td>
<td>16.66</td>
<td>2.776</td>
<td>0.78</td>
<td>0.589</td>
</tr>
<tr>
<td>N*Alpha</td>
<td>12</td>
<td>50.78</td>
<td>4.232</td>
<td>1.18</td>
<td>0.290</td>
</tr>
<tr>
<td>Beta*Pa</td>
<td>10</td>
<td>25.47</td>
<td>2.547</td>
<td>0.71</td>
<td>0.714</td>
</tr>
<tr>
<td>Beta*Alpha</td>
<td>20</td>
<td>91.98</td>
<td>4.599</td>
<td>1.29</td>
<td>0.178</td>
</tr>
<tr>
<td>Pa*Alpha</td>
<td>8</td>
<td>23.76</td>
<td>2.969</td>
<td>0.83</td>
<td>0.576</td>
</tr>
<tr>
<td>3-Way Interactions</td>
<td>154</td>
<td>501.66</td>
<td>3.258</td>
<td>0.91</td>
<td>0.770</td>
</tr>
<tr>
<td>N<em>Beta</em>Pa</td>
<td>30</td>
<td>124.12</td>
<td>4.137</td>
<td>1.16</td>
<td>0.257</td>
</tr>
<tr>
<td>N<em>Beta</em>Alpha</td>
<td>60</td>
<td>168.60</td>
<td>2.810</td>
<td>0.79</td>
<td>0.883</td>
</tr>
<tr>
<td>N<em>Pa</em>Alpha</td>
<td>24</td>
<td>100.03</td>
<td>4.168</td>
<td>1.16</td>
<td>0.264</td>
</tr>
<tr>
<td>Beta<em>Pa</em>Alpha</td>
<td>40</td>
<td>108.91</td>
<td>2.723</td>
<td>0.76</td>
<td>0.860</td>
</tr>
<tr>
<td>4-Way Interactions</td>
<td>120</td>
<td>358.66</td>
<td>2.989</td>
<td>0.84</td>
<td>0.897</td>
</tr>
<tr>
<td>N<em>Beta</em>Pa*Alpha</td>
<td>120</td>
<td>358.66</td>
<td>2.989</td>
<td>0.84</td>
<td>0.897</td>
</tr>
<tr>
<td>Error</td>
<td>1440</td>
<td>5152.00</td>
<td>3.578</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1799</td>
<td>6541.42</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In this paper, to identify the variable settings that minimize the makespan, a response optimizer has been employed. The optimization plot, presented in Fig. 5, shows the effect of each factor (columns) on the makespan. The red vertical lines on the graph represent the optimal factor settings; and, the blue horizontal line represents the makespan for the optimal factor level. The optimization plot clearly shows that the population size of 20, with $\beta = 3$, and $Pa = \alpha = 0.5$ produces the best performance for the set J120.

For the sake of simplicity, and according to our preliminary experiments, it can be concluded that N=10 to 20, $Pa = 0.5$, $\beta = 1.5$, and $\alpha = 0.5$ are sufficient even for the hardest test instances regardless of the benchmark set and the number of schedules. In addition, our simulation results show that the step size...
scaling factor, \( \alpha \), can be linked with the upper bounds (\( Ub \)) and lower bounds (\( Lb \)) in the following empirical way:

\[
\alpha \rightarrow \alpha (Ub - Lb)
\]  \( (9) \)

which makes that the steps are not too aggressive (jumping out of the feasible domain), thus ensuring most newly generated solutions in the right search regions. Here \( Ub \) and \( Lb \) are d-dimensional vectors with the same dimensions as the solution vector, representing priority values for activities. The results also imply that the convergence rate is often insensitive to the parameters used (refer to the p-values of the ANOVA table). It means that the fine adjustment is not needed for any given problem.

5. Computational Experiments

In this section, the results of our computational experiments are represented. Experiments of the CSA have been run under the Windows operating system on a VAIO PC with 1.8 GHz speed and 2GB RAM, and the algorithm has been coded in Visual C++. The benchmark instances have been selected from Kolisch and Sprecher (1997), consisting of four different sets, namely, J30, J60, J90, and J120, with projects of 30, 60, 90, and 120 non-dummy activities, respectively. The benchmark instances have been generated by the problem generator ProGen, which is available on www.om-db.wi.tum.de/psplib/main.html.

Each of the J30, J60, and J90 consists of 480 test instances, and the fourth set (J120) includes 600 instances. Each problem instance has four renewable resources, and every ten instances are comprising a group. Hence, each of the first three sets has only 48 groups of instances, while the set J120 has 60 groups. For each group, three parameters including network complexity, resource factor, and resource strength are varied, and instances are systematically generated. The network complexity is related to the successors of activities and shows the average number of the immediate successors of the activity \( i \). The resource factor reflects the average number of resources requested for each activity \( i \), and the resource strength presents the scarceness of the resource capacities. The optimal solution of the set J30 is available in the literature, unlike the set J60, J90, and J120 in which the optimal solutions of many instances are not available. Hence, for these sets, instead of the measure of percentage deviation from the optimal solution, i.e., \( \%Dev\text{-}Opt \), the measure of percentage deviation from the CPM (Critical Path Method) lower bound has been adopted, i.e., \( \%Dev\text{-}CPM \). As a regular measure of performance, the CPM lower bound is calculated by ignoring resource constraints and solving the simplified problem. Besides, since the results for the set J90 have not provided for the well-reported algorithms, we have ignored this set from the analysis.

To solve each benchmark instance, the limits of 1000, 5000, and 50,000 schedules are usually set for the intended algorithm. Setting a limit on the number of schedules makes it possible to compare the performance of the CSA with other algorithms, regardless of the speeds of computers on which they run.

Table 2 represents the results of the cuckoo search algorithm. The instance set and the number of schedules (as the stopping criteria) have been indicated in the first and second columns of the table, respectively. The third column represents the average percentage deviation from the lower bounds. As mentioned above, for set J30, the optimal values are known, so they use as the lower bound. However, the optimal solution values for the set J60, and J120 are not available. Hence, CPM is considered as the lower bound and the percentage deviation from the CPM is calculated for each test instance. The number of instances for which the proposed algorithm can obtain the best-known makespan in PSPLIB is illustrated in the fourth column as of January 2014. The obtained results of Table 2 reveals that the proposed algorithm is capable of finding the optimal solutions for all instances in the J30 problem set, with a limit of 50000 schedules.
Table 2. The obtained results for the benchmark instances

<table>
<thead>
<tr>
<th>Problem Sets</th>
<th>Schedules</th>
<th>Ave. LB dev. (%)</th>
<th># of Best</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1,000</td>
<td>0.06</td>
<td>467</td>
</tr>
<tr>
<td>J30</td>
<td>5,000</td>
<td>0.03</td>
<td>475</td>
</tr>
<tr>
<td></td>
<td>50,000</td>
<td>0.00</td>
<td>480</td>
</tr>
<tr>
<td>J60</td>
<td>1,000</td>
<td>11.48</td>
<td>385</td>
</tr>
<tr>
<td></td>
<td>5,000</td>
<td>10.88</td>
<td>405</td>
</tr>
<tr>
<td></td>
<td>50,000</td>
<td>10.62</td>
<td>456</td>
</tr>
<tr>
<td>J120</td>
<td>1,000</td>
<td>32.88</td>
<td>274</td>
</tr>
<tr>
<td></td>
<td>5,000</td>
<td>31.19</td>
<td>332</td>
</tr>
<tr>
<td></td>
<td>50,000</td>
<td>29.36</td>
<td>401</td>
</tr>
</tbody>
</table>

To compare the CSA to the other algorithms available for solving the resource constrained project scheduling problem, Tables 3, 4 and 5, show the results achieved by the best well-reported algorithms on the J30, J60, and J120, respectively. The basis of this comparison is the average deviation from the lower bounds (whether %Dev-CPM or %Dev-Opt). The following tables report the CSA performance concerning 1000, 5000, and 50,000 schedules.

As is clear, the proposed CSA has the best results for the J120 problem set and the second one for the J30 and J60 sets. Moreover, CSA outperforms most of the available metaheuristic algorithms in the literature. Kolisch and Hartmann define the non-dominated heuristic as an algorithm that has a higher average deviation than another algorithm for at least one combination of instances like J30, J60, or J120. According to this definition and owing to the obtained results on the set J120, we can conclude that CSA is a non-dominated algorithm and has a higher performance in analogy to the best available algorithms.

The thing that needs to be mentioned is that most of the metaheuristic optimization algorithms can converge quickly to the current best solution, but not necessarily to the global best solution. Hence, there is no guarantee for satisfying the global convergence condition. However, it has been proved that cuckoo search meets the global convergence requirements, and thus, it has ensured global convergence properties (Wang, et al., 2012).

Table 3. Comparison with the other procedures based on % Dev-OPTIMAL for the set J30.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th># of Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>Cuckoo Search Algorithm</td>
<td>This paper</td>
<td>0.06</td>
</tr>
<tr>
<td>Magnet-based GA</td>
<td>Zamani (2001)</td>
<td>0.14</td>
</tr>
<tr>
<td>Hybrid ACO/SS</td>
<td>Chen, et al. (2010)</td>
<td>0.14</td>
</tr>
<tr>
<td>Enhanced Scatter Search</td>
<td>Mobini, et al. (2009)</td>
<td>0.05</td>
</tr>
<tr>
<td>GA</td>
<td>Mendes, et al. (2009)</td>
<td>0.06</td>
</tr>
<tr>
<td>Hybrid GA</td>
<td>Valls, et al. (2008)</td>
<td>0.27</td>
</tr>
<tr>
<td>Scatter Search/electromagnetism</td>
<td>Debels, et al. (2006)</td>
<td>0.27</td>
</tr>
<tr>
<td>Double Justification</td>
<td>Valls, et al. (2005)</td>
<td>0.34</td>
</tr>
</tbody>
</table>
Table 4. Comparison with the other procedures based on %Dev-CPM for the set J60.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th># of Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>CSA</td>
<td>This paper</td>
<td>11.48</td>
</tr>
<tr>
<td>Magnet-based GA</td>
<td>Zamani (2001)</td>
<td>11.33</td>
</tr>
<tr>
<td>Hybrid GA</td>
<td>Valls, et al. (2008)</td>
<td>11.56</td>
</tr>
<tr>
<td>Scatter Search/electromagnetism</td>
<td>Debels, et al. (2006)</td>
<td>11.73</td>
</tr>
<tr>
<td>Double Justification</td>
<td>Valls, et al. (2005)</td>
<td>12.21</td>
</tr>
</tbody>
</table>

Table 5. Comparison with the other procedures based on %Dev-CPM for the set J120.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Reference</th>
<th># of Schedules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1,000</td>
</tr>
<tr>
<td>CSA</td>
<td>This paper</td>
<td>32.88</td>
</tr>
<tr>
<td>Magnet-based GA</td>
<td>Zamani (2001)</td>
<td>34.02</td>
</tr>
<tr>
<td>Hybrid ACO/SS</td>
<td>Chen, et al. (2010)</td>
<td>35.19</td>
</tr>
<tr>
<td>Enhanced Scatter Search</td>
<td>Mobini, et al. (2009)</td>
<td>34.51</td>
</tr>
<tr>
<td>GA</td>
<td>Mendes, et al. (2009)</td>
<td>35.87</td>
</tr>
<tr>
<td>Hybrid GA</td>
<td>Valls, et al. (2008)</td>
<td>34.07</td>
</tr>
<tr>
<td>Scatter Search/electromagnetism</td>
<td>Debels, et al. (2006)</td>
<td>35.22</td>
</tr>
<tr>
<td>Double Justification</td>
<td>Valls, et al. (2005)</td>
<td>35.39</td>
</tr>
</tbody>
</table>

6. Concluding Remarks and Future Research Avenues

In this paper, the cuckoo search algorithm has been developed for solving the RCPSP. To this end, a random key representation with a modified lower bound and upper bound has been employed as the encoding scheme. Besides, in constructing a schedule from the generated solutions, a procedure has been used which works between the serial and parallel schedule generation scheme, based on a switching parameter generated from the Uniform distribution. Updating procedure using Levy flights (rather than standard random walks) is the backbone of the CSA, but equally significant are the random local refinement and the improvement method. As Levy flights have infinite mean and variance, the CSA can explore the search space more efficiently than algorithms using standard Gaussian processes. This advantage has been combined with both local and global search capabilities and has guaranteed global convergence which makes the cuckoo search very efficient. In comparison with the best well-reported state-of-the-art algorithms, the computational results revealed that the CSA has the appropriate ability to solve the hardest problem instances of the RCPSP.
To identify the optimal parameter settings that minimize the makespan, a multi-level factorial design has been employed. Based on our preliminary experiments, regardless of the benchmark set and the limit of schedules, N=10 to 20, Pa = 0.5, β = 1.5, and α = 0.5 are sufficient even for the hardest test instances. In addition, results and analysis imply that the convergence rate, to some extent, is not sensitive to the parameters used (refer to the p-values of the ANOVA table). It means that the fine adjustment is not needed for any given problem.

**Funding**

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

**Conflicts of Interest**

The authors declare no conflict of interest in preparing this article.

**References**

• Ernst, B., Bloh, M., Seume, J. R. & Gómez González, A., 2012. Implementation of the cuckoo search algorithm to optimize the design of wind turbine rotor blades. Denmark, s.n.


This work is licensed under a Creative Commons Attribution 4.0 International License.