Information Sciences 277 (2014) 680–693

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem

Georgios Koulinas *, Lazaros Kotsikas, Konstantinos Anagnostopoulos

Department of Production and Management Engineering, Democritus University of Thrace, 12 Vas. Sofias st., 671 00 Xanthi, Greece

ARTICLE INFO

Article history:
Received 24 December 2012
Received in revised form 21 January 2014
Accepted 26 February 2014
Available online 21 March 2014

Keywords:
Particle swarm optimization
Hyper-heuristics
Project scheduling
Resource constrained project scheduling problem

ABSTRACT

In this paper, we propose a particle swarm optimization (PSO) based hyper-heuristic algorithm for solving the resource constrained project scheduling problem (RCPSP). To the best of our knowledge, this is the first attempt to develop a PSO hyper-heuristic and apply to the classic RCPSP. The hyper-heuristic works as an upper-level algorithm that controls several low-level heuristics which operate to the solution space. The solution representation is based on random keys. Active schedules are constructed by the serial scheduling generation scheme using the priorities of the activities which are modified by the low-level heuristics of the algorithm. Also, the double justification operator, i.e. a forward–backward improvement procedure, is applied to all solutions. The proposed approach was tested on a set of standard problem instances of the well-known library PSPLIB and compared with other approaches from the literature. The promising computational results validate the effectiveness of the proposed approach.

1. Introduction

The resource constrained project scheduling problem (RCPSP) has been recognized by researchers and practitioners as one of the most important and challenging problems in operational research. Many other problems in production planning, grid computing, network packet switching, and facility location involve the scheduling concept given that these problems commonly accompany the cost objective related to certain constraints. The objective of the RCPSP is to minimize the total project makespan by scheduling the activities of a project such that both the precedence relations between the activities and limited resource availabilities are satisfied.

Traditionally, the RCPSP is a problem that appears growing interest by the researchers. Exact methods, such as branch and bound algorithms [22,10,49] have been proposed in the literature for solving the problem to optimality. However, treating instances with large number of activities requires considerable computation time since it has been proven that the RCPSP is an NP-hard problem in the strong sense as a generalization of the job shop scheduling problem [8]. Thus, heuristic and meta-heuristic approaches have been proposed to find near-optimum solutions in reasonable computational time and overcome the combinatorial explosion phenomenon.

Simple heuristics based on priority rules were proposed by Kolisch [42]. Many of these priority rules are borrowed from machine scheduling, and particularly from job shop scheduling. A review and computational analysis of priority rules can be found in Klein [39]. Bouleimen and Lecocq [9] proposed simulated annealing, Nonobe and Ibaraki [51] and Klein [40]

* Corresponding author. Tel.: +30 2541079332; fax: +30 2541079343.
E-mail addresses: gkoulina@pme.duth.gr (G. Koulinas), lkotsikas@yahoo.gr (L. Kotsikas), kanagn@civil.duth.gr (K. Anagnostopoulos).

http://dx.doi.org/10.1016/j.ins.2014.02.155
0020-0255/© 2014 Elsevier Inc. All rights reserved.

developed tabu search, and Palpant et al. [53] and Lova et al. [46] presented variable neighborhood search and hybrid metaheuristics, respectively.

As for the population based approaches, genetic algorithms [31,3,34,32,61,47,28,54], scatter search [21], and ant systems [48] have also been proposed. Kochetov and Stolyar [41] developed an evolutionary algorithm which combines genetic algorithm, path relinking, and tabu search. Recently, Chen et al. [17] proposed a hybrid ant colony and scatter search algorithm, and Wang and Fang [62] developed a hybrid estimation of distribution algorithm to solve the RCPSP.

New approaches such as neurogenetic algorithms [1], artificial immune algorithms [50], bee algorithms [69], and shuffled frog leaping algorithms [26] are applied recently to the RCPSP.

It is clear from the literature that the research interest in RCPSP and its extensions is great and constant over the years, confirming that these problems are a popular research field for applying the latest optimization techniques.

Despite the success of heuristics and metaheuristics in solving real world problems, they are not easily applicable to new problems, or even new instances of similar problems. This drawback arises from the range of parameter choices involved when using these approaches since there are no commonly accepted directions on parameter tuning. As a result, developing such a problem-specific method is expensive to develop and maintain. The core idea of hyper-heuristics is to develop algorithms that are more generally applicable than many of the classic search approaches. A hyper-heuristic, attempts to find a good performing sequence of heuristics in a given problem instance rather than trying to solve the problem directly. Therefore, the goal is to develop generic methods, which should produce solutions of acceptable quality, based on a set of easy-to-implement low-level heuristics. A hyper-heuristic is employed as a higher level methodology, and given a particular problem instance or class of instances, manages a number of low-level heuristics producing combinations of algorithmic applications to find a near-optimal solution.

Over the last few years there is a growing literature in the field of hyper-heuristics. Storer et al. [56], Dowsland et al. [25], and Anagnostopoulos and Koulinas [5] proposed simulated annealing hyper-heuristics, Burke et al. [12] developed tabu search and Qu and Burke [55] proposed variable neighborhood search. Additionally, population based hyper-heuristics have been developed. Dorndorf and Pesch [24], Hart and Ross [30], Han and Kendall [29], and Anagnostopoulos and Koulinas [6] proposed genetic algorithm based hyper-heuristics. Gascón-Moreno et al. [27] proposed an evolutionary hyper-heuristic. Burke et al. [13] proposed ant colony based algorithm, Anagnostopoulos and Koulinas [7] introduced a GRASP-based hyper-heuristic algorithm, and Koulinas and Anagnostopoulos [45] proposed a threshold accepting hyper-heuristic. Recently, Crawford et al. [19], Alinia Ahandani et al. [4], and Ahmed et al. [2] proposed hyper-heuristic approaches based on particle swarm optimization.

Numerous applications have also been developed. In particular, hyper-heuristic algorithms have been proposed for treating well-known scheduling problems such as the flow shop scheduling problem [52], and the job shop scheduling problem [24,56,29,64]. Recent reviews of hyper-heuristics and their applications could be found in [16,14].

Recently, hyper-heuristics have been applied for handling project scheduling problems. Anagnostopoulos and Koulinas [5] proposed a simulated annealing hyper-heuristic for the resource leveling problem, Anagnostopoulos and Koulinas [6] developed a genetic hyper-heuristic algorithm for the RCPSP, Anagnostopoulos and Koulinas [7] introduced a GRASP hyper-heuristic for RCPSP, and Koulinas and Anagnostopoulos [45] proposed a threshold accepting hyper-heuristic for both resource allocation and leveling. These approaches are developed within a widely used commercial project management software to enhance its efficiency and proved to be of acceptable efficiency.

Hyper-heuristics have been recently classified by Burke et al. [15] and a new definition of the term "hyper-heuristic" has been proposed. A hyper-heuristic algorithm is defined as "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems". From this definition, two categories of hyper-heuristics are distinguished: heuristic selection and heuristic generation. Heuristic selection refers to methods for choosing or selecting existing heuristics and heuristic generation refers to methods for generating new heuristics from components of existing heuristics. An extensive analysis of this newly proposed classification can be found in Burke et al. [15].

In this paper, we propose a particle swarm optimization based hyper-heuristic algorithm (PSO-HH) to handle the resource constrained project scheduling problem. The proposed algorithm manages solution methods rather than solutions, and employs simple low-level heuristics. The procedure has been developed using Visual Basic 2010 programming language.

2. Formulation of the problem

A project is represented by an acyclic activity-on-node (AoN) network topology. The network consists of n activities (nodes) and precedence relations (arcs) between activities, and R renewable resources. We denote d_i and f_i the non-preemptable duration and the finish time of an activity i. Nodes 1 and n are dummy activities representing the start and finish of the project with no ingoing and outgoing arcs, respectively. The RCPSP is defined as follows:

$$\min f_n \tag{1}$$

subject to

$$f_i \leqslant f_j - d_j \quad \text{for all precedence relations } (i,j) \tag{2}$$

$$f_1 = 0, \quad d_1 = 0, \quad d_n = 0 \tag{3}$$

$$\sum_{i \in P_t} u_{irt} \leqslant U_r \quad \text{for } t = 1, \ldots, f_n \text{ and } r \in R \tag{4}$$

The objective is to find a feasible schedule so that the project duration f_n to be minimized (1). Eq. (2) makes sure that the precedence relations are satisfied, and Eq. (3) that the project starts at time instant 0. The usage of renewable resources that are assigned to each activity is assumed to remain constant throughout the progress of the activity. Also there is resource availability U considered, constant through the project execution, which cannot be violated at any time period, i.e., the sum u_{irt} of resource type r usage of all ongoing activities P_t in time period t does not exceed the resource availability U_r (Eq. (4)). Fig. 1 illustrates an example of an RCPSP instance with a corresponding feasible schedule.

In this definition of the classic problem, it is expected that due to the limited resource units that can be assigned to activities, the project makespan resulting after resource constrained scheduling will most likely be larger than the initial duration calculated by CPM, without taking into account the resource usage limit.

The RCPSP was starting point for confronting various real world decision making issues. Therefore, many extensions of the basic model have been proposed, for instance by introducing generalized precedence relations for the activities (RCPSP-GPR), allowing for activity pre-emption (PRCPSP), and considering other types of objective functions (e.g. resource leveling problem-RLP). A detailed presentation and an extensive literature review for the basic RCPSP and its extensions can be found in [23,33,63].

3. PSO hyper-heuristic algorithm

3.1. The PSO metaheuristic

The particle swarm optimization (PSO) metaheuristic is an evolutionary approach proposed by Kennedy and Eberhart [38]. It is a multi-agent algorithm, inspired from the collective behavior in decentralized systems, and has been applied for solving complex optimization problems.

The PSO concept considers a population of randomly positioned particles, which is called a swarm, and searches for the best position with best fitness. The swarm consists of individual agents named particles, and represented by vectors, where each particle corresponds to a potential solution of an optimization problem. A particle is treated as a point in a multidimensional space and the status of a particle is characterized by its position and velocity [37]. The position of every particle

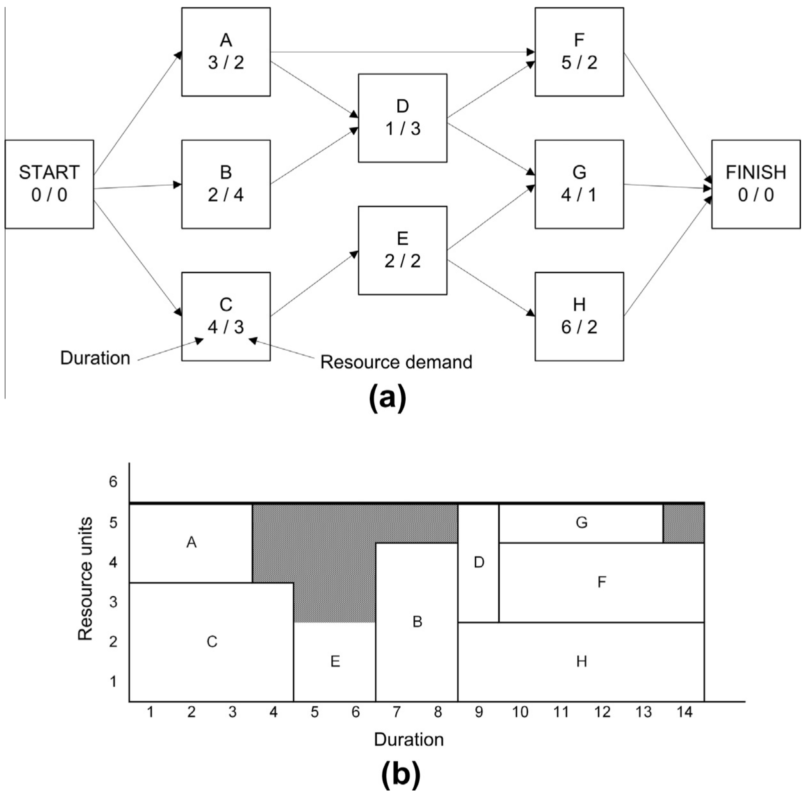

Fig. 1. (a) An example project instance for the RCPSP and (b) a feasible solution.

represents a candidate solution to the problem at hand. In each iteration, every particle moves to a new position within the search space guided by a variable velocity which is a vector. The corresponding fitness of the new position of each particle is calculated and the best position found is memorized. As a result, the velocity is a critical parameter in searching for a good solution. There are two different positions used for updating the velocity. The first is the best experienced position of all particles which memorizes the global best solution obtained by all agents. The second is the best position that a single particle encountered, which memorizes the individual experience of a particle. A swarm of particles moves in the trajectory which is updated using the following formulas [37]:

$$V_i(t) = w(t) \times V_i(t-1) + c_1 \times r_1 \times (X_i^L - X_i(t-1)) + c_2 \times r_2 \times (X^G - X_i(t-1)) \tag{5}$$

$$X_i(t) = V_i(t) + X_i(t-1) \tag{6}$$

where the particles are represented by $i = 1, \ldots, M$ with M meaning the total number of the particles in a swarm, and the iterations by $t = 1, \ldots, T$ with T meaning the maximum number of iterations. The $X_i(t) = \{x_{i1}(t), \ldots, x_{iN}(t)\}$ is the position of the ith particle in the tth iteration consisted of N components, and $V_i(t) = \{v_{i1}(t), \ldots, v_{iN}(t)\}$ is the velocity (N-dimensional) of the particle i in iteration t. The $X_i^L = \{x_{i1}^L, \ldots, x_{iN}^L\}$ represents the individual-local best (L_{best}) position that the particle i found so far, while $X^G = \{x_{i1}^G, \ldots, x_{iN}^G\}$ denotes the global best (G_{best}) position found by the swarm. The c_1 and c_2 are positive constants used as learning factors indicating how the ith particle approaches either the individual or the global experience position, while r_1 and r_2 are random numbers uniformly distributed between 0 and 1 influencing the tradeoff between the global exploitation and local exploration abilities during search. In addition, $w(t)$is the inertia weight for controlling the impact of the previous velocities on the new velocity.

The efficiency of the PSO depends on its ability to perform global search of the search space and local search as well. Larger influence of the global best position on the particles causes faster convergence of the algorithm towards specific points, and so the PSO emphasizes in the exploitation of the search space. On the other hand, settings that facilitate local search, prevent the swarm from getting trapped in local optima, and the PSO emphasizes in the exploration of the search space.

Particle swarm optimization metaheuristics have been recently used for treating project scheduling problems. Jarboui et al. [35] solved the multi-mode case of RCPSP, Czogalla and Fink [20] investigated the application of PSO in combination with different population topologies in comparison to state-of-the-art methods from the literature. Kemmoe-Tchomte and Gourgand [37] proposed an extension of the PSO system that integrates a new displacement of the particles (the balance between the intensification process and the diversification process). Zhang et al. [67] proposed a permutation based PSO for the RCPSP, and Zhang and Li [68] employed multi-objective particle swarm optimization for construction time–cost tradeoff problems and Chen [18] used PSO with justification for the RCPSP. Also, PSO used for parameter tuning of hyperheuristics [19] and for product design and manufacturing [66,65].

3.2. The concept of hyper-heuristics

Hyper-heuristic algorithms have been recently appeared a growing research effort as effective search techniques aiming to provide solutions of a good quality for many optimization problems. Hyper-heuristics are multi-level/multi-layer algorithms that manage a set of heuristic operators or generate low-level heuristics from existing operators [14].

Hyper-heuristics are similar to metaheuristic algorithms in the sense that can be applied to a variety of problems. However, the core idea of a metaheuristic is operating directly on the space of solutions. Hyper-heuristics search the solution space indirectly, using the low-level heuristics. As a low-level heuristic could be considered already known heuristic operators, simple metaheuristic algorithms or even a group of other hyper-heuristics. Each low-level heuristic can search the solution space, modify the current solution and find a new one, which is evaluated. A hyper-heuristic does not search for a good solution to the problem itself but selects at each iteration of the process a proper low-level heuristic or combination of heuristics for improving the solution. Since a hyper-heuristic is employed as the upper algorithmic layer that manages some underlying heuristics [11], it does not affect the manner that each low-level heuristic works. However, it has access to problem specific information such as the value of the objective function.

3.3. The proposed PSO-HH hyper-heuristic

A particle is a vector of eight integer numbers each representing a low-level heuristic algorithm. The particles impose the order that the low-level heuristics are applied to the solution domain. The algorithm evaluates the individuals using as fitness function the Eq. (1). After preliminary experiments it was found that a swarm of 20 particles, which corresponds to a number of $20 \times 8 = 160$ low-level heuristics' applications, and consequently schedules constructed, is suitable for the proposed algorithm.

The sequence that the low-levels applied and found the best solution for every particle is stored. Both the best duration and the heuristics sequence (namely the solution) are the individual experience of a particle used for updating the velocity. Also, the best experienced position of all particles, which is the global best solution obtained by all particles, is used.

The flowchart and the pseudocode of the proposed process are illustrated in Fig. 2 and in Table 1, respectively. For a project of N activities that is represented by activity-on-node network topology, the procedure to implement the PSO-HH for the

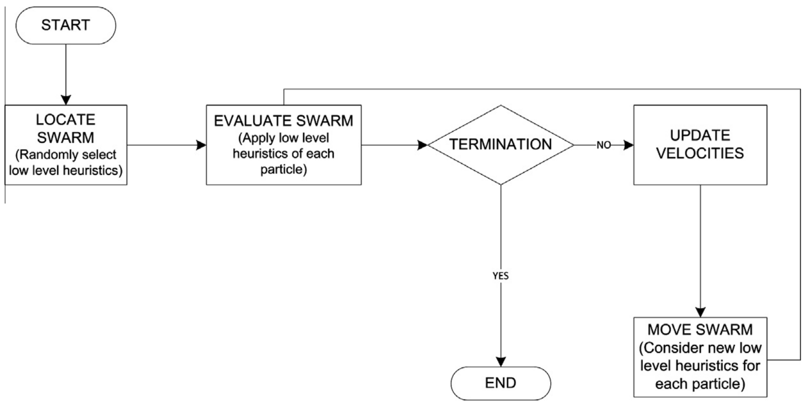
START
LOCATE SWARM (Randomly select low level heuristics)
EVALUATE SWARM (Apply low level heuristics of each particle)
TERMINATION
NO
YES
UPDATE VELOCITIES
MOVE SWARM (Consider new low level heuristics for each particle)
END

Fig. 2. The flowchart of the proposed PSO-HH.

Table 1
Pseudocode of the PSO-HH.

```
Step 1: **Start** procedure PSO_HH. Initialize the position vector of each particle of the swarm. Define $r_1$, $r_2$, $c_1$, $c_2$, $w(t)$
Step 2: **Evaluate** Swarm
  For each particle $i$ do
    For $j = 1$ to 8
      **Call** the $j_{th}$ low level heuristic of the particle to the solution space
      **Apply** the Serial Schedule Generation Scheme
      **Apply** the Double Justification operator
      **Update** the particle and the swarm experience $(X_i^L, X^G)$, and the
      best found solution and as well
    Next $j$
  Next $i$
Step 3: **Check** the stopping condition. If it is not met go to Step 4. Otherwise go to Step 6
Step 4: **Calculate** the new velocities $V_i(t)$ for each particle $i$
Step 5: **Move** the particles to their new positions. Go to Step 2
Step 6: **Finish** the procedure PSO_HH
```

RCPSP works as follows. A swarm with M particles is considered. For the ith particle, $i = (1, \ldots, M)$, its position consists of $j = 1, \ldots, K$ components $X_i = (X_{i1}, \ldots, X_{iK})$, where X_{ij} is the jth component of the position.

In the first step of the process (Locate Swarm) we set the iteration counter equal to zero and initialize the M particles by randomly selecting a heuristic rule, for each component of particles' position. The initial swarm is created in such a manner that each heuristic appears once in each particle. Also, the initial vectors of priorities include randomly generated numbers from the interval $(0,1)$. Initially the global solution is empty, and so we set each element of the local best particles equal to the initial particles of the swarm. Then, the initial global best solution is equal to the best located particle of the local best particles.

In the next step, the evaluation of the swarm is performed. The low-level heuristics of each particle are consecutively applied on the solution space modifying the priorities of the activities. Each vector of priorities corresponds to a feasible schedule constructed using the serial scheduling generation scheme (see Section 3.5). In addition, we apply the well-known double justification procedure [61] to the obtained schedule in order to achieve local improvement (see Section 3.6). The serial scheduling generation scheme and the justification operator are applied, and thus the objective function is evaluated for each low-level heuristic application and the best found solution for every particle is memorized.

Subsequently, the velocities are calculated using Eq. (5). Owing to the fact that learning factors c_1 and c_2 lead to little difference in the PSO performance [57], the value 2 is selected for both c_1 and c_2. The r_1 and r_2 are random numbers within the interval $(0,1)$. The inertia weight for controlling the impact of the previous velocities on the new velocity $(w(t))$ is considered after trial experiments to be equal to 1.

Based on the updated velocities, each particle moves to a new position (step Move Swarm). The new position is calculated using Eq. (6). The output of Eq. (6) is a set of real values. We get an integer for each x_{ij} with Eq. (7)

$$x_{ij} = \left\{ \begin{array}{ll} 1 & \text{if } x_{ij} < 1 \\ l_h & \text{if } x_{ij} > l_h \\ \text{int}(x_{ij}) & \text{otherwise} \end{array} \right\} \tag{7}$$

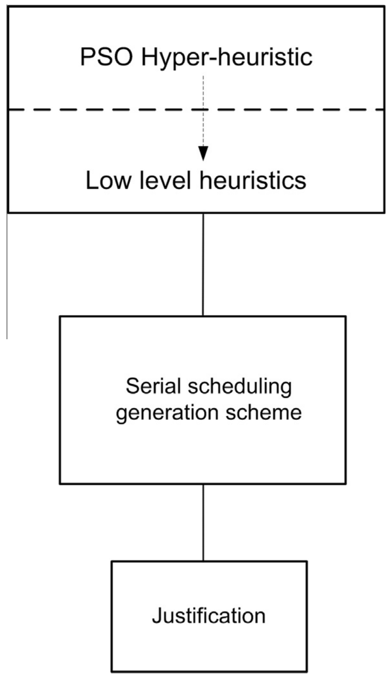

Fig. 3. An outline of the proposed approach function.

where $i = (1, \ldots, M)$, $j = 1, \ldots, K$ and l_h counts the number of the available low-level heuristics. The PSO is terminated when the iteration counter is equal to the maximum total number of iterations. Otherwise the process returns to the "Evaluate Swarm" step and the next hyper-heuristic iteration of the search starts.

Fig. 3 reviews the main components of the proposed algorithm. The PSO-HH is modifying the sequence in that low-level heuristics are applied to the solution space. After using a low-level and considering the vector of priorities, the serial scheduling generation scheme is used to construct a feasible schedule and compute the makespan. Finally, the double justification procedure is used to apply local search and possibly improve the resulted makespan.

In order to validate the functionality of the proposed hyper-heuristic under the chosen algorithmic parameters settings, we have applied a preliminary analysis in which the convergence of the algorithm is studied. Three network instances (one for each set) have been selected randomly (networks J3021_5, J606_8, J12040_4), and the convergence history of the algorithm is compared. Figs. 4a, 5a, and 6a illustrate the algorithmic convergence curves for each network. It is monitored the best found duration by the PSO-HH and the average duration for all particles.

The curves indicate that the parameter settings are suitable for the algorithm since it converges relatively quickly, and the total number of iterations proved to be large enough to allow the hyper-heuristic to effectively treat the given problem.

In addition, the initial random position of each particle is illustrated in the radar charts (Figs. 4b, 5b and 6b), so as the particles' movements until the best solution is found. It shows that the manner that particles use to move to better positions assists the relatively quick convergence to the optimal (J30) or the best known (J60 and J120) solution. Also, it is observed that the initial position of each particle strongly affects its efficiency.

3.3.1. Low-level heuristics

The basic hyper-heuristic concept emphasize on the use of simple and easy-to-implement low-level heuristics. Also, the design of the set of the problem-specific low-level heuristics is important for the efficiency of the applied hyper-heuristic. The proposed PSO-HH controls eight low-level heuristics which are simple heuristic operators modifying priority values of project activities. Conforming to the core hyper-heuristic idea, the proposed heuristics are based on simple moves such as "swap" and "replace". A solution that low-level heuristics operate on, is encoded using a vector of priority values for each activity excluding the dummy activities representing the start and the finish of a project. The element pr_{ih} of the priorities vector $PR_i = (pr_{i1}, \ldots, pr_{iN})$ expresses the priority of the hth activity of the project in the scheduling process. After the modification of the current vector by a low-level heuristic, the conversion to standardized random key (SRK) format (see Section 3.4) is applied and the serial scheduling generation scheme constructs the corresponding schedule.

We partition the low-level heuristics into three subsets according to how they modify the priorities' vector. The first group (L1–L2) includes shifting heuristics, while the second group (L3–L5) contains heuristics which replace and swap priorities. The third group (L6–L8) includes crossover-based heuristics. A brief description of the low-level heuristics follows:

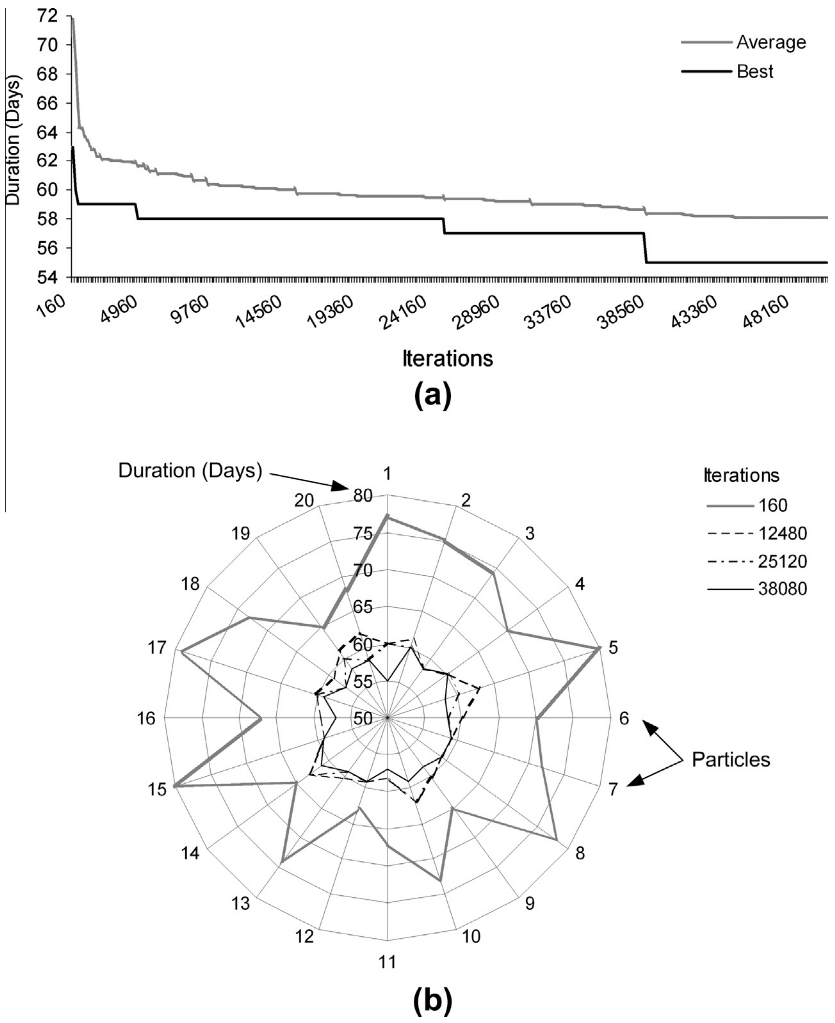

Fig. 4. Network J3021_5 – (a) convergence curves for best found and average durations of all particles and (b) convergence of each particle during the search until the best solution found.

- L1 (Adjacent values, right shift): Define a subset (sequence) of priorities of length l within the array RK, where $l \in [1, N-1]$ is randomly chosen and N is the array RK length. Move this subset to a randomly chosen position situated between the end of the subset and the end of the array (right shift), and the other involved values so as to fill the empty positions (Fig. 7). In case that the end of the subset coincides with the end of the array the heuristic is repeated.
- L2 (Adjacent values, left shift): Same as L1 but the subset is moved in a random position situated between the start of the subset and the start of the array (left shift). The heuristic is repeated if the start of the subset coincides with the start of the array.
- L3 (Replace with a higher value): Select at random an activity and replace its priority value with a higher one randomly chosen.
- L4 (Replace with a lower value): Select at random an activity and replace its priority value with a lower one randomly chosen.
- L5 (Swap two values): Select at random two activities and swap their priority values.
- L6 (One point crossover): Consider the currently best found priority array as first parent and a randomly created one as second parent. Create an offspring array of priorities by the one point crossover operator (Fig. 8).
- L7 (Two point crossover): Best found priority array and a randomly created are used as parents to create an offspring array of priorities by the classic two points crossover operator.
- L8 (Parameterized uniform crossover): As in the previous heuristics, the best found vector of priorities and a vector with random values are used as parents. For each gene of the two parents, a real random number in the interval [0, 1] is generated. If this number is smaller than a predefined index (c_p), then the low-level heuristic of the first parent is copied to the offspring's gene. Otherwise, the gene of the second parent passes to the new gene. After initial experimentation the c_p parameter has been fixed to 0.6 (Fig. 9).

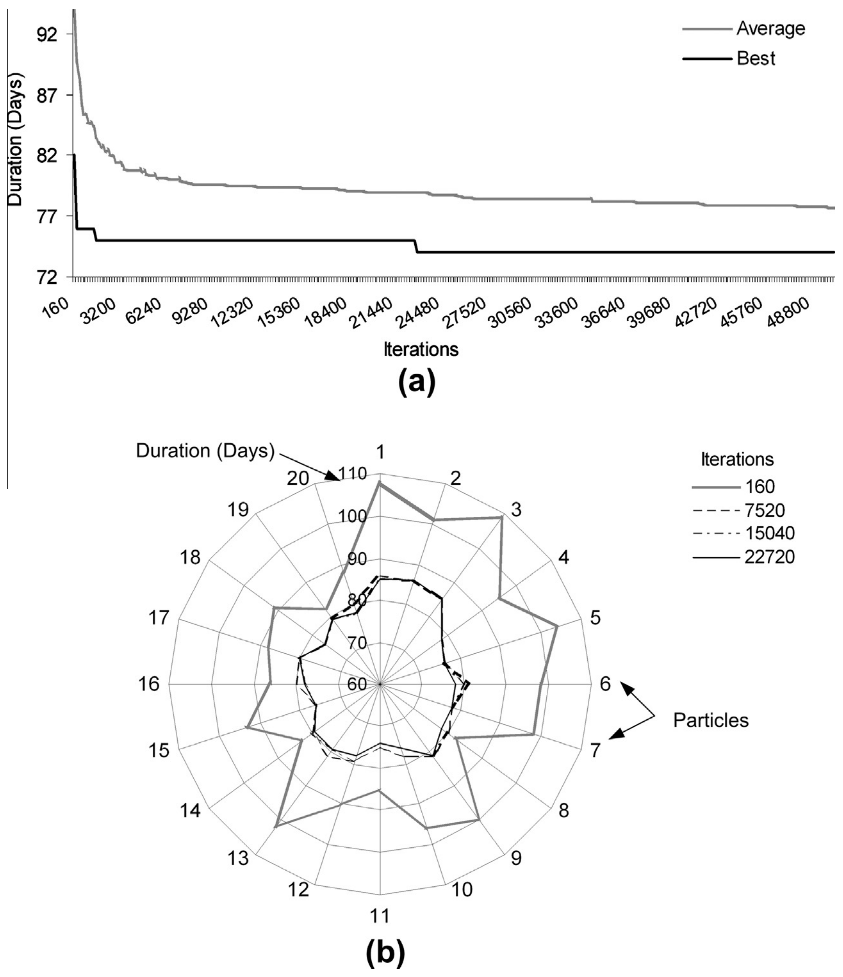

Fig. 5. Network J606_8 – (a) convergence curves for best found and average durations of all particles and (b) convergence of each particle during the search until the best solution found.

3.4. Solution representation scheme

Typically, a heuristic for the RCPSP does not operate directly on the schedules, but on some structures representing a schedule that is appropriate for applying the algorithm. The representation schemes that are most widely used in the literature are the activity-list (AL) representation and the random-key (RK) representation which both use the idea of activities prioritization. In the AL representation, a sequence of the activities is used where the position of an activity in this sequence determines its relative priority compared to the other activities. A solution (schedule) is represented by a permutation vector of the activities satisfying the precedence constraints meaning that each activity is positioned in the list after all its predecessors. To construct the corresponding schedule, the list is decoded with the serial scheduling generation scheme (Section 3.5) where the activities are selected according to their order in the list and scheduled as soon as possible in the partial schedule.

In the RK representation a vector of real numbers between 0 and 1 is used such that the *i*th vector element works as a priority value for the *i*th activity. Using the serial schedule generation scheme, these priority values can be used to construct an active schedule by scheduling each activity as early as possible without violating the precedence and resource constraints.

Both the AL and the RK schemes have the characteristic that there exist many different representations for one single schedule. In order to deal with this drawback, we include to our procedure the process proposed by Debels et al. [21] to convert the RK vector into the so-called standardized random-key (SRK) representation vector.

More precisely, we first rank the activities according to their start times in the schedule, and then replace their priority values by the place in the ranking. By using the SRK representation, each schedule is uniquely associated with an RK-vector. However, after a move in the neighbourhood search by a low-level heuristic, the newly generated priority vectors are possibly not conform to the SRK format. Thus, each new priority vector needs to be transformed into the SRK form, while evaluating the objective function value.

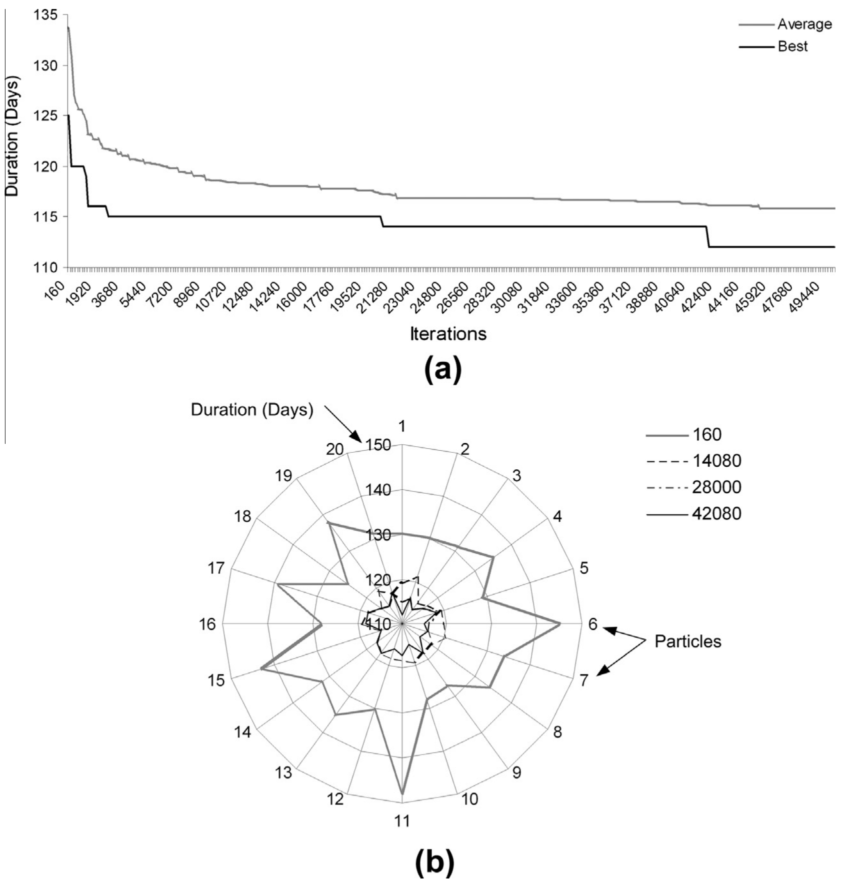

Fig. 6. Network J12040_4 – (a) convergence curves for best found and average durations of all particles and (b) convergence of each particle during the search until the best solution found.

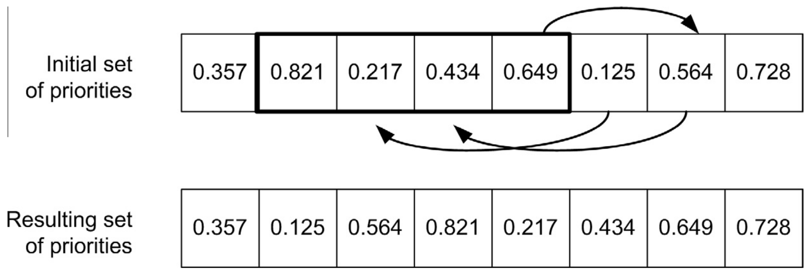

Fig. 7. The adjacent values right shift low-level heuristic (L1).

3.5. Serial scheduling generation scheme

There are two different schedule generation schemes (SGS) available in the literature; the serial SGS (SSGS) which is activity oriented and the parallel SGS (PSGS) that is time oriented. The difference between the serial and parallel schemes is that the SSGS constructs active schedules, while the parallel generates non-delay schedules. Non-delay schedules are schedules where idle times are introduced only when feasible activity available does not exist, and it is not guaranteed that they include an optimal solution. Thus the set of non-delay schedules is a subset of the set of active schedules. As a result, the PSGS may miss an optimal schedule. It is known from the literature [23] that any schedule that is generated by the SSGS belongs to the set of active schedules which contains at least one optimal solution, meaning that there is a sequence of choices that can be made by the SSGS and leads to an optimal schedule. Owing to this great advantage over PSGS, in the proposed approach we use the SSGS as a decoding function.

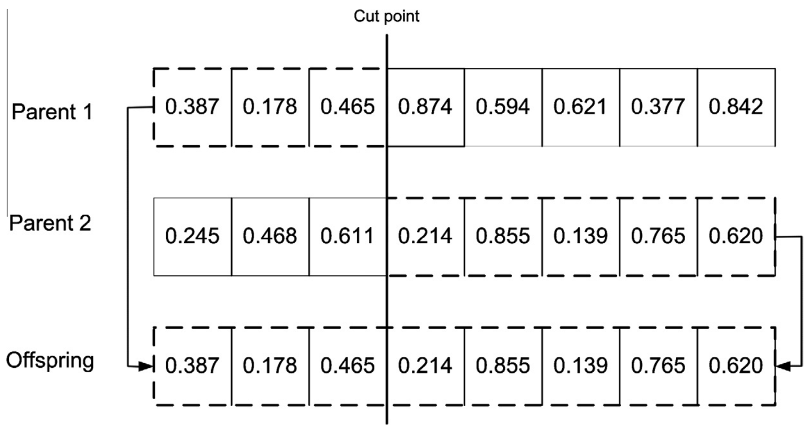

Fig. 8. The one point crossover low-level heuristic (L6).

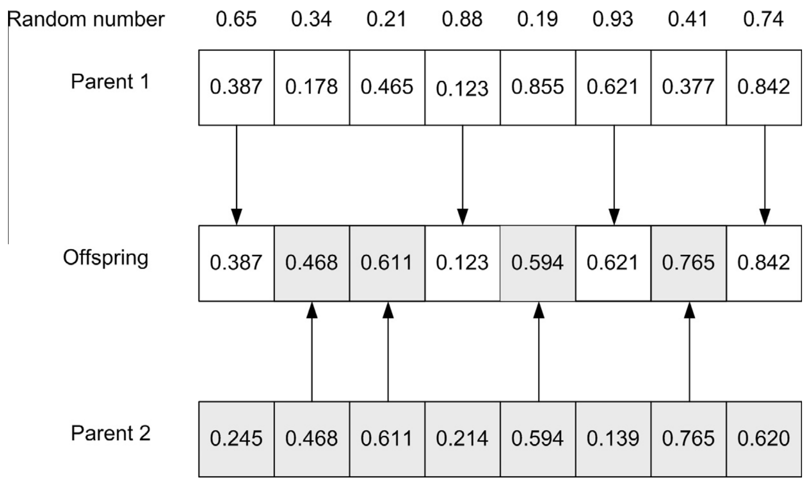

Fig. 9. The parameterized uniform crossover low-level heuristic (L8) with $c_p = 0.6$.

The SSGS was proposed by Kelley [36]. The SSGS can be incorporated with an ordered list of all activities so it can choose the first activity in the list that is feasible. It sequentially adds activities to the schedule until a feasible complete schedule is obtained. In each iteration, the next activity in the priority list is chosen and for that activity the first as soon as possible starting time is assigned such that no precedence or resource constraint is being violated.

The SSGS starts with a partial schedule that contains only the dummy start activity starting at time 0. Then, the complete schedule is constructed in n iterations. In addition, the set of the eligible activities $E(t)$ is considered for each iteration t i.e., the activities that have not been scheduled and all its immediate predecessors have already been scheduled. At each stage, one activity j is selected from the set of available activities and added to the partial schedule.

3.6. Double justification

Double justification [59] is a simple local search technique which is incorporated into many RCPSP algorithms. It is applied to schedules constructed by a scheduling scheme to improve the solution quality. It works by shifting all activities within a schedule to the right and then to the left in order to obtain a better schedule. This method proved to be effective since it produces improvements in a very small computational time cost [59]. Generally, we apply the double justification operator to each generated schedule of the PSO-HH. Double justification is similar [44] to the forward–backward improvement procedure proposed by Tormos and Lova [58] and enhanced by the study of Tormos and Lova [59].

4. Computational analysis

We applied the PSO-HH to the standard instance sets from PSPLIB (http://129.187.106.231/psplib/) [43]. The sets J30 and J60 consist of 480 problem instances with four resource types and 30 and 60 activities, respectively. The set J120 consists of

Table 2
Average deviations from optimal solutions for the J30 set of instances.

Algorithm	Reference	Number of schedules		
		1000	5000	50,000
PSO-HH	Our procedure	0.26	0.04	0.01
GA-TS-Path relinking	Kochetov and Stolyar [41]	0.10	0.04	0.00
SS-EM	Debels et al. [21]	0.27	0.11	0.01
Hybrid GA	Valls et al. [61]	0.27	0.06	0.02
GA-FBI	Valls et al. [60]	0.34	0.20	0.02
GA	Alcaraz and Maroto [3]	0.33	0.12	NA
TS	Nonobe and Ibaraki [51]	0.46	0.16	0.05
Sampling-LFT, FBI	Tormos and Lova [58]	0.30	0.16	0.07
GA	Hartmann [32]	0.38	0.22	0.08
GA-activity list	Hartmann [31]	0.54	0.25	0.08
Sampling-LFT, FBI	Tormos and Lova [59]	0.30	0.17	0.09
TS	Klein [40]	0.42	0.17	NA
Sampling-random, FBI	Valls et al. [60]	0.46	0.28	0.11
SA	Bouleimen and Lecocq [9]	0.38	0.23	NA
GAPS	Mendes et al. [47]	0.06	0.02	0.01
ACOSS	Chen et al. [17]	0.14	0.06	0.01
Neurogenetic (FBI)	Agarwal et al. [1]	0.13	0.10	NA
Art. Imm. Alg.	Mobini et al. [50]	0.05	0.03	0.00
GA-FBI	Goncalves et al. [28]	0.32	0.02	0.01
HEDA	Wang and Fang [62]	0.38	0.14	NA
SFL	Fang and Wang [26]	0.36	0.21	0.18
JPSO	Chen [18]	0.29	0.14	0.04
BA-FBI	Ziarati et al. [69]	0.42	0.19	0.04
GANS	Proon and Jin [54]	1.83	1.27	0.71

Table 3
Average deviations from critical path lower bound for the J60 set of instances.

Algorithm	Reference	Number of schedules		
		1000	5000	50,000
PSO-HH	Our procedure	11.74	11.13	10.68
GA-TS-Path relinking	Kochetov and Stolyar [41]	11.71	11.17	10.74
SS-EM	Debels et al. [21]	11.73	11.10	10.71
Hybrid GA	Valls et al. [61]	11.56	11.10	10.73
GA-FBI	Valls et al. [60]	12.21	11.27	10.74
GA	Alcaraz and Maroto [3]	NA	NA	NA
TS	Nonobe and Ibaraki [51]	12.97	12.18	11.58
Sampling-LFT, FBI	Tormos and Lova [58]	12.18	11.87	11.54
GA	Hartmann [32]	12.21	11.70	11.21
GA-activity list	Hartmann [31]	12.68	11.89	11.23
Sampling-LFT, FBI	Tormos and Lova [59]	12.14	11.82	11.47
TS	Klein [40]	12.77	12.03	NA
Sampling-random, FBI	Valls et al. [60]	NA	NA	NA
SA	Bouleimen and Lecocq [9]	12.75	11.90	NA
GAPS	Mendes et al. [47]	11.72	11.04	10.67
ACOSS	Chen et al. [17]	11.75	10.98	10.67
Neurogenetic (FBI)	Agarwal et al. [1]	11.51	11.29	NA
Art. Imm. Alg.	Mobini et al. [50]	11.17	10.80	10.55
GA-FBI	Goncalves et al. [28]	NA	11.56	10.57
HEDA	Wang and Fang [62]	11.97	11.43	NA
SFL	Fang and Wang [26]	11.44	10.87	10.66
JPSO	Chen [18]	12.03	11.43	11.00
BA-FBI	Ziarati et al. [69]	12.55	12.04	11.16
GANS	Proon and Jin [54]	11.35	10.53	10.52

600 instances with four resource type and 120 activities. Also, the optimal or the best known solutions achieved by algorithms proposed in the literature are available in the above website. As for J30, the optimal makespan for each instance is known. In J60 and J120 sets, the currently best found solutions (upper bounds) and lower bounds are provided. The lower bounds are defined by the critical path duration in the resource relaxation of the problem using the classic CPM.

The efficiency of the proposed algorithm is compared with 23 heuristic algorithms for the RCPSP that are either reported in the study of Kolisch and Hartmann [43] or recently proposed in the literature. For achieving a machine-independent comparison, we defined the number of generated and evaluated schedules in PSO-HH to 1000, 5000, and 50,000, since these are

Table 4
Average deviations from critical path lower bound for the J120 set of instances.

Algorithm	Reference	Number of schedules		
		1000	5000	50,000
PSO-HH	Our procedure	35.20	32.59	31.23
GA-TS-Path relinking	Kochetov and Stolyar [41]	34.74	33.36	32.06
SS-EM	Debels et al. [21]	35.22	33.10	31.57
Hybrid GA	Valls et al. [61]	34.07	32.54	31.24
GA-FBI	Valls et al. [60]	35.39	33.24	31.58
GA	Alcaraz and Maroto [3]	39.36	36.57	NA
TS	Nonobe and Ibaraki [51]	40.86	37.88	35.85
Sampling-LFT, FBI	Tormos and Lova [58]	36.49	35.81	35.01
GA	Hartmann [32]	37.19	35.39	33.21
GA-activity list	Hartmann [31]	39.37	36.74	34.03
Sampling-LFT, FBI	Tormos and Lova [59]	36.24	35.56	34.77
TS	Klein [40]	NA	NA	NA
Sampling-random, FBI	Valls et al. [60]	35.39	33.24	31.58
SA	Bouleimen and Lecocq [9]	42.81	37.68	NA
GAPS	Mendes et al. [47]	35.87	33.03	31.44
ACOSS	Chen et al. [17]	35.19	32.48	30.56
Neurogenetic (FBI)	Agarwal et al. [1]	34.65	34.15	NA
Art. Imm. Alg.	Mobini et al. [50]	34.01	32.57	31.48
GA-FBI	Goncalves et al. [28]	NA	35.94	32.76
HEDA	Wang and Fang [62]	35.44	33.61	NA
SFL	Fang and Wang [26]	34.83	33.20	31.11
JPSO	Chen [18]	35.71	33.88	32.89
BA-FBI	Ziarati et al. [69]	37.72	36.76	34.55
GANS	Proon and Jin [54]	33.45	31.51	30.45

Table 5
Average computational time for the instance sets.

Instance set	Iterations	Avg. time (s)
J30	1000	0.23
	5000	1.15
	50,000	11.43
J60	1000	0.39
	5000	2.00
	50,000	20.65
J120	1000	1.48
	5000	7.99
	50,000	81.96

the mostly used limitations in the literature. PSO-HH was implemented in Visual Basic 2010 and all the experiments performed on a PC Pentium IV with 3.00 GHz CPU and 512 MB RAM running under the Windows XP operating system. In this analysis, we used the parameter settings that appeared in Section 3.3.

4.1. Experimental results

Table 2 presents the percentage deviations from the optimal makespan for the instance set J30 in which all problem instances have been solved to optimality. The PSO-HH hyper-heuristic solved 425, 470, and 476 out of 480 problems to optimality for 1000, 5000, and 50,000 schedules with an average deviation from the optimal solution of 0.26%, 0.04%, and 0.01% respectively. The hyper-heuristic achieves the 6th, 4th, and 3rd best performance, respectively.

For J60, and J120 instance sets, the optimal solutions are not known, so the results are reported in terms of average percentage deviation from the critical path based lower bound. Table 3 summarizes the results for J60 test instances. The proposed procedure achieved the best found solution in 391, 406, and 431 of the total 480 instances; and the average deviation from the critical path based lower bound is 11.74%, 11.13%, and 10.68% for 1000, 5000, and 50,000 schedules, which ranks PSO-HH performance 9th, 8th, and 7th respectively.

Table 4 displays the results for J120 set. The hyper-heuristic achieved a makespan equal to the currently best found for 212, 246, and 272 out of 600 instances, ranked 8th, 5th, and 4th, with average deviations of 35.20%, 32.59%, and 31.23% respectively, for 1000, 5000, and 50,000 schedules.

The results show that PSO-HH proved to be effective on treating these problem sets since it achieves near-optimal solutions for the instance set J30, and can produce competitive solutions for the larger J60 and J120 problem instance sets. The hyper-heuristic is not dominated by any other algorithm in this comparison. Note that, as in the well known study of Kolisch and Hartmann [43], the concept of dominance is used to determine the best heuristics. In line with that, a heuristic $h1$ is dominated by a heuristic $h2$ if $h1$ has for at least one combination of instance set and number of generated schedules a higher average deviation than $h2$ without having for any of the other combinations a lower average deviation. The proposed algorithm appeared not very efficient for 1000 iterations confirming that a small number of schedules generated is not enough in order to show the benefits arising from the different heuristic search methods cooperation.

Table 5 shows the average computational effort in seconds that the hyper-heuristic spent for treating the problem instance sets. The average CPU time cost of PSO-HH is acceptable, and increases almost linearly with respect to the number of schedules for each set. Also, the results show an increase in the processing time required when the problem size increases, meaning that larger scale problems are more difficult to solve.

5. Conclusion

In this paper we proposed a new PSO based hyper-heuristic algorithm to address a well-known and very challenging scheduling problem, the classic resource constrained project scheduling problem (RCPSP). To the best of our knowledge, this is the first PSO hyper-heuristic appeared in the literature for solving RCPSP, and the first hyper-heuristic that treated the well-known problem instance library PSPLIB. The computational analysis illustrates that the results are very competitive with other approaches used in the literature.

The hyper-heuristic is the main component of a flexible multi-level procedure that does not need to know how the low-level heuristics operate to the solution space but it only requires knowledge of the objective function and its value. This characteristic of the hyper-heuristic framework facilitates the development of problem-independent hyper-heuristic algorithms that share a common pool of low-level heuristics for project scheduling problems. Consequently, we believe that the flexibility of hyper-heuristics and the promising results reported in this study should encourage researchers to further develop hyper-heuristics for treating project scheduling problems.

References

[1] A. Agarwal, S. Colak, S. Erenguc, A neurogenetic approach for the resource-constrained project scheduling problem, Comput. Oper. Res. 38 (1) (2011) 44–50.

[2] A. Ahmed, M. Ali, A. Sajid, A.H.S. Bukhari, Particle swarm based hyper-heuristic for tackling real world examinations scheduling problem, Aust. J. Basic Appl. Sci. 5 (10) (2011) 1406–1413.

[3] J. Alcaraz, C. Maroto, A robust genetic algorithm for resource allocation in project scheduling, Ann. Oper. Res. 102 (1–4) (2001) 83–109.

[4] M. Alinia Ahandani, M.T. Vakil Baghmisheh, M.A. Badamchi Zadeh, S. Ghaemi, Hybrid particle swarm optimization transplanted into a hyper-heuristic structure for solving examination timetabling problem, Swarm Evol. Comput. 7 (December) (2012) 21–34.

[5] K.P. Anagnostopoulos, G.K. Koulinas, A simulated annealing hyperheuristic for construction resource levelling, Constr. Manage. Econ. 28 (2) (2010) 163–175.

[6] K.P. Anagnostopoulos, G.K. Koulinas, A genetic hyperheuristic algorithm for the resource constrained project scheduling problem, in: Proceedings of the 2010 IEEE Congress on Evolutionary Computation, CEC 2010, IEEE Computer Society Press, Barcelona, Spain, 2010.

[7] K. Anagnostopoulos, G. Koulinas, Resource-constrained critical path scheduling by a GRASP based hyperheuristic, ASCE J. Comput. Civil Eng. 26 (2) (2012) 204–213.

[8] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. 5 (1) (1983) 11–24.

[9] K. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, Eur. J. Oper. Res. 149 (2) (2003) 268–281.

[10] P. Brucker, S. Knust, A. Schoo, O. Thiele, A branch-and-bound algorithm for the resource constrained project scheduling problem, Eur. J. Oper. Res. 107 (2) (1998) 272–288.

[11] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, Hyper-heuristics: an emerging direction in modern search technology, in: F. Glover, G.A. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Dordrecht, 2003, pp. 457–474.

[12] E. Burke, G. Kendall, E. Soubeiga, A tabu search hyperheuristic for timetabling and rostering, J. Heuristics 9 (6) (2003) 451–470.

[13] E.K. Burke, G. Kendall, D. Landa-Silva, R. O'Brien, E. Soubeiga, An ant algorithm hyper-heuristic for the project presentation scheduling problem, in: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, CEC 2005, IEEE Computer Society Press, Edinburgh, Scotland, 2005, pp. 2263–2270.

[14] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, Hyper-Heuristics: A Survey of the State of the Art, School of Computer Science and Information Technology, University of Nottingham, Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, 2010. <http://www.cs.nott.ac.uk/~gxo/papers/hhsurvey.pdf>.

[15] E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. Woodward, A classification of hyper-heuristic approaches, in: M. Gendreau, J-Y. Potvin (Eds.), Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol. 146, Springer, 2010, pp. 449–468.

[16] K. Chakhlevitch, P. Cowling, Hyperheuristics: recent developments, in: C. Cotta, M. Sevaux, K. Sorensen (Eds.), Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, vol. 136, Springer, Berlin, 2008, pp. 3–29.

[17] W. Chen, Y.-j. Shi, H.-f. Teng, X.-p. Lan, L-c. Hu, An efficient hybrid algorithm for resource-constrained project scheduling, Inf. Sci. 180 (6) (2010) 1031–1039.

[18] R.M. Chen, Particle swarm optimization with justification and designed mechanisms for resource-constrained project scheduling problem, Expert Syst. Appl. 38 (6) (2011) 7102–7111.

[19] B. Crawford, R. Soto, E. Monfroy, W. Palma, C. Castro, F. Paredes, Parameter tuning of a choice-function based hyperheuristic using Particle Swarm Optimization, Expert Syst. Appl. 40 (5) (2013) 1690–1695.

[20] J. Czogalla, A. Fink, Particle swarm topologies for resource constrained project scheduling, in: N. Krasnogor et al. (Eds.), Nature Inspired Cooperative Strategies for Optimization, Springer-Verlag, Berlin, Heildeberg, 2009, pp. 61–73.

[21] D. Debels, B. De Reyck, R. Leus, M. Vanhoucke, A hybrid scatter search/electromagnetism metaheuristic for project scheduling, Eur. J. Oper. Res. 169 (2) (2006) 638–653.

[22] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Manage. Sci. 38 (12) (1992) 1803–1818.

[23] E.L. Demeulemeester, W.S. Herroelen, Project Scheduling – A Research Handbook, Kluwer Academic Publishers, Boston, 2002.

[24] U. Dorndorf, E. Pesch, Evolution based learning in a job shop scheduling environment, Comput. Oper. Res. 22 (1) (1995) 25–40.
[25] K.A. Dowsland, E. Soubeiga, E. Burke, A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation, Eur. J. Oper. Res. 179 (3) (2007) 759–774.
[26] C. Fang, L. Wang, An effective shuffled frog-leaping algorithm for resource-constrained project scheduling problem, Comput. Oper. Res. 39 (5) (2012) 890–901.
[27] J. Gascón-Moreno, S. Salcedo-Sanz, B. Saavedra-Moreno, L. Carro-Calvo, A. Portilla-Figueras, An evolutionary-based hyper-heuristic approach for optimal construction of group method of data handling networks, Inf. Sci. 247 (2013) 94–108.
[28] J.F. Goncalves, M.G.C. Resende, J.J.M. Mendes, A biased random-key genetic algorithm with forward–backward improvement for the resource constrained project scheduling problem, J. Heuristics 17 (5) (2011) 467–486.
[29] L. Han, G. Kendall, An investigation of a tabu assisted hyper-heuristic genetic algorithm, in: Proceedings of the 2003 IEEE Congress on Evolutionary Computation, CEC 2003, IEEE Computer Society Press, Canberra, Australia, 2003, pp. 2230–2237.
[30] E. Hart, P. Ross, A heuristic combination method for solving job-shop scheduling problems, in: A.E. Eiben, T. Back, M. Schoenauer, H.P. Schwefel (Eds.), Parallel Problem Solving from Nature – PPSN V, Lecture Notes in Computer Science, vol. 1498, Springer, Heidelberg, 1998, pp. 845–854.
[31] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Nav. Res. Log. 45 (7) (1998) 733–750.
[32] S. Hartmann, A self-adapting genetic algorithm for project scheduling under resource constraints, Nav. Res. Log. 49 (5) (2002) 433–448.
[33] S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource constrained project scheduling problem, Eur. J. Oper. Res. 207 (1) (2010) 1–14.
[34] K.S. Hindi, H. Yang, K. Fleszar, An evolutionary algorithm for resource-constrained project scheduling, IEEE Trans. Evol. Comput. 6 (5) (2002) 512–518.
[35] B. Jarboui, N. Damak, P. Siarry, A. Rebai, A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems, Appl. Math. Comput. 195 (1) (2008) 299–308.
[36] J.E. Kelley, The critical-path method: resources planning and scheduling, in: J.F. Muth, G.L. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, NJ, 1963, pp. 347–365.
[37] S. Kemmoe-Tchomte, M. Gourgand, Particle swarm optimization: a study of particle displacement for solving continuous and combinatorial optimization problems, Int. J. Prod. Econ. 121 (1) (2009) 57–67.
[38] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the Fourth IEEE Conference on Neural Networks, Perth, Australia, 1995, pp. 1942–1948.
[39] R. Klein, Scheduling of Resource Constrained Projects, Kluwer Academic Publishers, Boston, 2000.
[40] R. Klein, Project scheduling with time-varying resource constraints, Int. J. Prod. Res. 38 (16) (2000) 3937–3952.
[41] Y. Kochetov, A. Stolyar, Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, in: Proceedings of the 3rd International Workshop of Computer Science and Information Technologies, Russia, 2003.
[42] R. Kolisch, Efficient priority rules for the resource-constrained project scheduling problem, J. Oper. Manage. 14 (3) (1996) 179–192.
[43] R. Kolisch, A. Sprecher, PSPLIB – a project scheduling problem library, Eur. J. Oper. Res. 96 (1) (1997) 205–216.
[44] R. Kolisch, S. Hartmann, Experimental investigation of heuristics for resource-constrained project scheduling: an update, Eur. J. Oper. Res. 174 (1) (2006) 23–37.
[45] G.K. Koulinas, K.P. Anagnostopoulos, Construction resource allocation and leveling using a threshold accepting based hyperheuristic algorithm, ASCE J. Constr. Eng. Manage. 138 (7) (2012) 854–863.
[46] A. Lova, P. Tormos, M. Cervantes, F. Barber, An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes, Int. J. Prod. Econ. 117 (2) (2009) 302–316.
[47] J.J.M. Mendes, J.F. Goncalves, M.G.C. Resende, A random key based genetic algorithm for the resource constrained project scheduling problem, Comput. Oper. Res. 36 (1) (2009) 92–109.
[48] D. Merkle, M. Middendorf, H. Schmeck, Ant colony optimization for resource-constrained project scheduling, IEEE Trans. Evol. Comput. 6 (4) (2002) 333–346.
[49] A. Mingozzi, V. Maniezzo, S. Ricciardelli, L. Bianco, An exact algorithm for the resource constrained project scheduling problem based on a new mathematical formulation, Manage. Sci. 44 (5) (1998) 714–729.
[50] M. Mobini, Z. Mobini, M. Rabbani, An Artificial Immune Algorithm for the project scheduling problem under resource constraints, Appl. Soft Comput. 11 (2) (2011) 1975–1982.
[51] K. Nonobe, T. Ibaraki, Formulation and tabu search algorithm for the resource constrained project scheduling problem, in: C.C. Ribeiro, P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Publishers, Dordrecht, 2002, pp. 557–588.
[52] D. Ouelhadj, S. Petrovic, A cooperative hyper-heuristic search framework, J. Heuristics 16 (6) (2010) 835–857.
[53] M. Palpant, C. Artigues, P. Michelon, LSSPER: solving the resource-constrained project scheduling problem with large neighbourhood search, Ann. Oper. Res. 131 (1–4) (2004) 237–257.
[54] S. Proon, M. Jin, A genetic algorithm with neighborhood search for the resource-constrained project scheduling problem, Nav. Res. Log. 58 (2) (2011) 73–82.
[55] R. Qu, E.K. Burke, Hybridizations within a graph-based hyper-heuristic framework for university timetabling problems, J. Oper. Res. Soc. 60 (9) (2009) 1273–1285.
[56] R.H. Storer, S.D. Wu, R. Vaccari, Problem and heuristic search space strategies for job shop scheduling, INFORMS J. Comput. 7 (4) (1995) 453–467.
[57] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Inf. Process. Lett. 85 (6) (2003) 317–325.
[58] P. Tormos, A. Lova, A competitive heuristic solution technique for resource-constrained project scheduling, Ann. Oper. Res. 102 (1–4) (2001) 65–81.
[59] P. Tormos, A. Lova, An efficient multi-pass heuristic for project scheduling with constrained resources, Int. J. Prod. Res. 41 (5) (2003) 1071–1086.
[60] V. Valls, F. Ballestin, S. Quintanilla, Justification and RCPSP: a technique that pays, Eur. J. Oper. Res. 165 (2) (2005) 375–386.
[61] V. Valls, F. Ballestin, M.S. Quintanilla, A hybrid genetic algorithm for the RCPSP, Eur. J. Oper. Res. 185 (2) (2008) 495–508.
[62] L. Wang, C. Fang, A hybrid estimation of distribution algorithm for solving the resource-constrained project scheduling problem, Expert Syst. Appl. 39 (3) (2012) 2451–2460.
[63] J. Weglarz, J. Jozefowska, M. Mika, G. Waligora, Project scheduling with finite or infinite number of activity processing modes – a survey, Eur. J. Oper. Res. 208 (3) (2011) 177–205.
[64] A.S. Xanthopoulos, D.E. Koulouriotis, V.D. Tourassis, D.M. Emiris, Intelligent controllers for bi-objective dynamic scheduling on a single machine with sequence-dependent setups, Appl. Soft Comput. 13 (12) (2013) 4704–4717.
[65] A.R. Yildiz, K.N. Solanki, Multi-objective optimization of vehicle crashworthiness using a new particle swarm based approach, Int. J. Adv. Manuf. Technol. 59 (1–4) (2012) 367–376.
[66] A.R. Yildiz, A novel particle swarm optimization approach for product design and manufacturing, Int. J. Adv. Manuf. Technol. 40 (5–6) (2009) 617–628.
[67] H. Zhang, H. Li, C.M. Tam, Permutation-based particle swarm optimization for resource-constrained project scheduling, J. Comput. Civil Eng. 20 (2) (2006) 141–149.
[68] H. Zhang, H. Li, Multi-objective particle swarm optimization for construction time–cost tradeoff problems, Constr. Manage. Econ. 28 (1) (2010) 75–88.
[69] K. Ziarati, R. Akbari, V. Zeighami, On the performance of bee algorithms for resource-constrained project scheduling problem, Appl. Soft Comput. 11 (4) (2011) 3720–3733.