RAIRO-Oper. Res. 53 (2019) 1309–1330
https://doi.org/10.1051/ro/2019012

RAIRO Operations Research
www.rairo-ro.org

AN INNOVATIVE FOUR-LAYER HEURISTIC FOR SCHEDULING MULTI-MODE PROJECTS UNDER MULTIPLE RESOURCE CONSTRAINS

Reza Zamani[1,*]

Abstract. In this paper, an innovative four-layer heuristic is presented for scheduling multi-mode projects under multiple resource constraints. For this purpose, a biased-random sampling technique, a local search, a decomposition method, and an evolutionary search mechanism, each in a separate layer, are combined, with each layer passing its output to the next layer for improvement. The procedure has been designed based on the fact that what makes the scheduling of multi-mode projects hard to solve is a massive search space of modes compounded with the starting times of activities. That is why the procedure is aimed at balancing exploration *versus* exploitation in searching a massive search space. On the one hand, it exploits promising areas further and, on the other hand, it searches unexplored areas for expanding its range. Since the first layer provides an initial solution, and each of the other three layers can either improve the result of its previous layer or keep it unchanged, solutions never deteriorate and hence promising areas are exploited. Moreover, unexplored areas are searched effectively because each layer explores solution space differently than its previous layer. Based on whether or not an improvement each layer can make to the result of its previous layer, the effect of the corresponding layer on the performance of the procedure has been measured.

Mathematics Subject Classification. 90B50, 90B35, 90B40, 90.08, 68R05.

Received February 19, 2018. Accepted January 11, 2019.

1. Introduction

The Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP) is among of the most practical and hardest-to-solve problems in scheduling. The problem is involved with scheduling a project composed of several activities numbered from 0 to $n+1$. The two fictitious activities 0 and $n+1$, with the duration of zero and no resource requirements, demonstrate the starting and the ending of the project, respectively. Activities should be executed without any interruption, and require some scarce resources during their execution. These resources are divided into three groups, namely (i) renewable, (ii) non- renewable, and (iii) doubly-constrained [37].

Whereas renewable resources are renewed from period to period, non-renewable resources are fixed at the starting of the project and consumed by activities. An example of renewable resources is a machine or manpower, and an example for non-renewable resources is raw material. Money and other kinds of capital resources are doubly-constrained, in the sense that they are partly renewable and partly non-renewable. For instance, whereas money for the entire project is fixed like a non-renewable resource, it can become available as limited cash follow

Keywords. Multi-mode recourse-constrained projects, Scheduling, heuristics, combinatorial optimization.

[1] School of Computing and Information Technology, Wollongong University, NSW 2522 Wollongong, Australia.
*Corresponding author: reza@uow.edu.au

on a period-by-period basis, making it similar to renewable resources. In effect, if a resource has both a constant availability per period and a total limit, then it is considered as a doubly-constrained resource. That is why each doubly-constrained resource can be represented with one renewable and one non-renewable resource.

Deciding whether or not a resource is renewable depends on the circumstances which determine if the resource is renewed in a period-by-period basis. This means that whereas in a particular project, capital is renewable, in another project it can be non-renewable, and still in a third project, it can be a doubly-constrained resource.

Activity i can start only when all of its predecessors, which are shown by set P_i, have been completed, and with respect to each resource type, its required resources are available. There are R^r and R^e types of renewable and non-renewable resources, respectively, with the availability of renewable and non-renewable resources k and l being equal to a_k^r and a_l^e, respectively.

Moreover, there are M_i mode for activity i and when executed in mode m, its duration will be d_i^m. In this case, the activity i will require $r_{i,m,k}^r$ and $r_{i,m,l}^e$ from renewable and non-renewable resources k and l, respectively. Pre-emption is not allowed, in the sense that once an activity has started in a particular mode, it should be completed in the same mode without any interruption. The goal is to minimize the project duration by finding a mode and starting time for each activity.

Initially introduced by Elmaghraby [15], the MRCPSP was not originally involved with non-renewable resources and only renewable resources were present. More than a decade later, however, in the mathematical formulation presented in [44], non-renewable resources were introduced, extending the problem to an advanced version.

To distinguish between the two versions in the literature, in line with [50], whereas the original problem is shown with MRCPSP-R, with R standing for renewable resources, the problems which include both renewable and non-renewable resources are shown with MRCPSP. Figure 1 shows a sample multi-mode project and Figure 2 shows its optimal schedule along with the usage of renewable and non-renewable resources.

Subsuming the single mode resource-constrained project scheduling (RCPSP) problem, the MRCPSP compounds all the complexity of this subsumed problem, which itself is NP-hard [5]. Two other facts further demonstrating the hardness of the MRCPSP are as follows. First, the job shop scheduling problem, which itself is subsumed by the RCPSP, is NP-hard [28].

Second, in the MRCPSP with more than one non-renewable resources, even replacing the requirement of constructing an optimal solution with generating a feasible solution does not change the hardness of the problem, in the sense that finding a feasible solution is still NP-hard [25].

It is worth noting that since obtaining a feasible solution for the MRCPSP is an NP-hard problem, searching for modes which observe non-renewable resource constraints can be extremely time-consuming and this can make even some heuristics for the MRCPSP impractical. This hardness of the problem, on the one hand, and its great applicability on the other hand, has made it a vehicle for testing many combinatorial optimization techniques. In [50], two main ingredients of a variety of metaheuristics for the MRCPSP have been considered as (i) employing a solution representation, and (ii) using a schedule generation scheme.

With respect to schedule generation schemes, they considered serial and parallel methods, concerning mode representation they have distinguished mode vector (MV) and mode list (ML). The difference between MV and ML is that whereas the ith position of MV shows the mode of the activity in the ith position of the activity list, the ith position of ML shows the mode of the ith activity. In effect, in the MRCPSP, MV should always be accompanied by AL.

This paper presents an innovative four-layer heuristic, called the Local-search Evolutionary Decomposition-based Procedure (LEDP), for solving the MRCPSP. In the LEDP, a biased-random sampling technique, a local search, a decomposition technique, and an evolutionary search mechanism are combined to produce high-quality solutions. These four components work together to search the solution space effectively, each working in a separate layer and each layer passing its output to the next layer to improve.

First, a biased-random sampling technique, in the first layer, generates initial solutions. Towards improving the solution generated in the first layer, a local search in the second layer uses two right- and left-shift moves for each activity. The employed local search first keeps the modes of activities untouched and alters the orders of

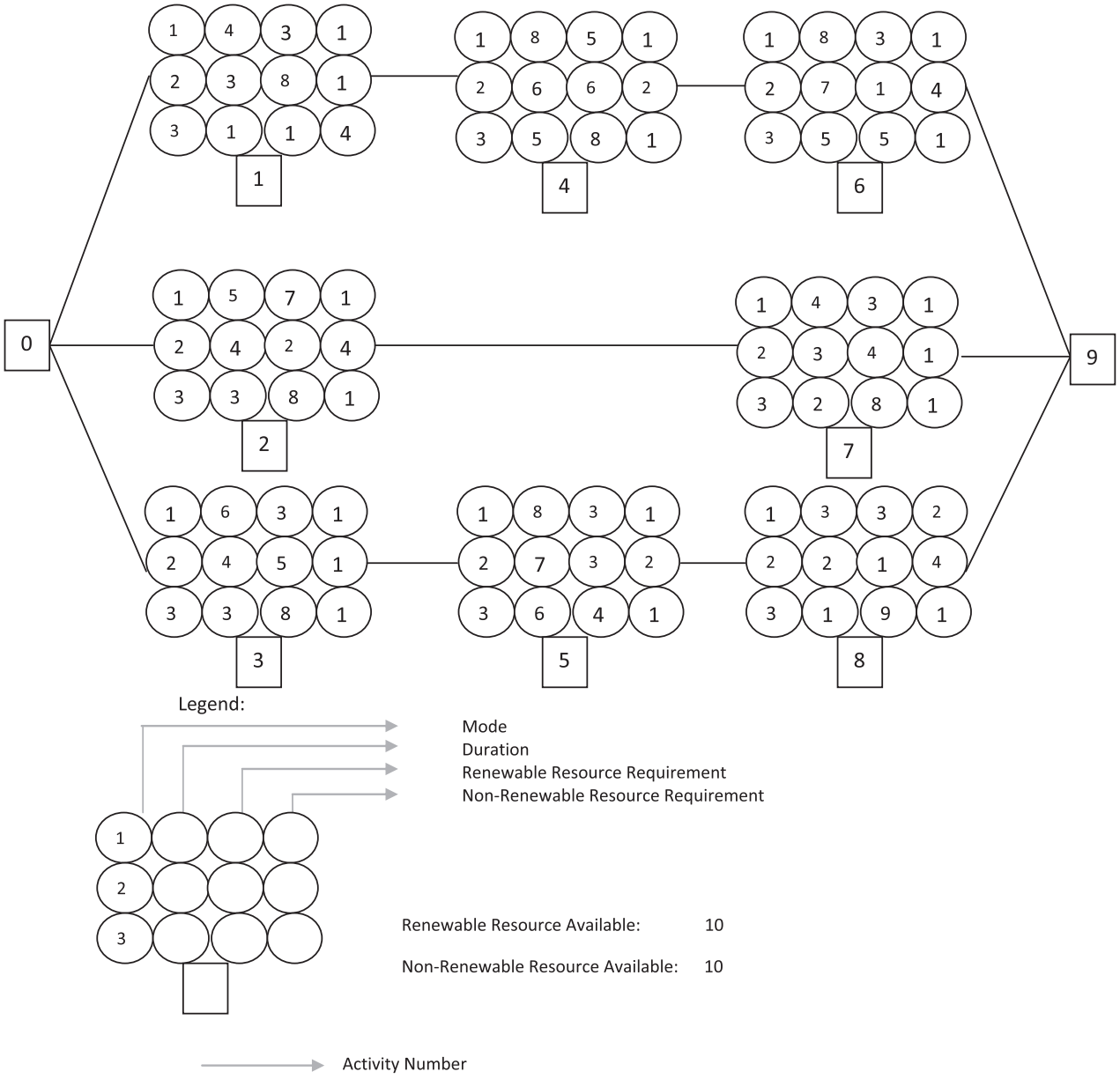
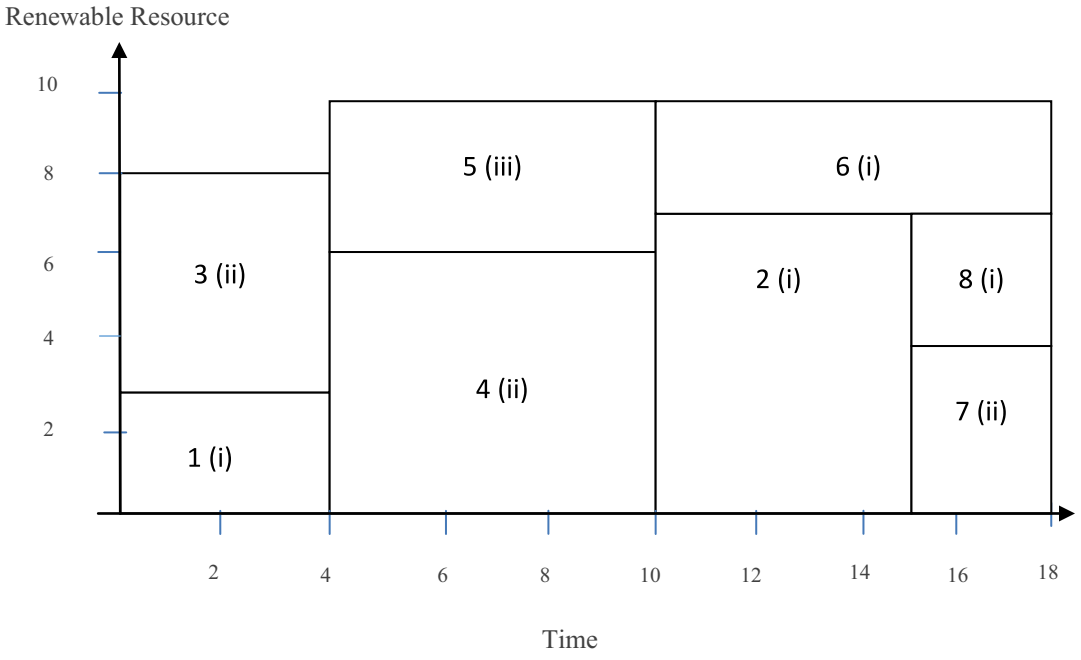


FIGURE 1. A Sample Multimode Project in which the duration and resource requirements of each activity have been shown based on the mode in which the activity is executed.

activities. When no improvement through such alternations becomes possible, modes are started to alter. The process of the right- and left-shift moves as well as mode moves continues until no improvement to the solution is possible.

Using the solution produced by the local search, the decomposition technique, in the third layer, without changing modes only alters the starting times of activities to improve the result further. For this purpose, it



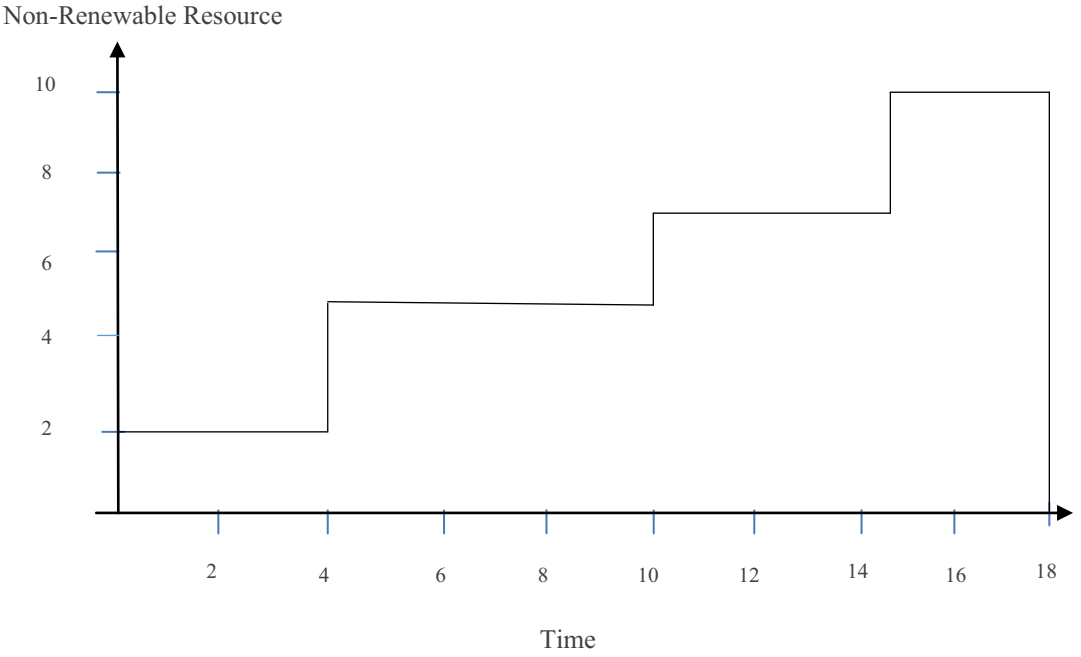


FIGURE 2. The optimal schedule of the sample project along with the usage of renewable as well as non-renewable resources.

decomposes a project into smaller sub-projects and schedules the decomposed sub-projects to optimality within a time-limit of several milliseconds, if possible.

Further decomposition for a sub-project is performed when such a time-limit does not allow the sub-project to be scheduled to optimality. In the hope of further improvement, the evolutionary search technique, in the fourth layer, searches the vicinity of the solution obtained by decomposition technique. For this purpose, it first fills a pool of slightly perturbed copies of the solution obtained in the third layer, and then, towards its possible improvement, alters both modes and orders of activities with crossover and mutation operations.

The rest of the paper is as follows. Section 2 presents the related work. The LEDP is discussed in Sections 3 and 4 is devoted to the results of the computational experiments. Concluding remarks are presented in Section 5, which also discusses future research direction for the LEDP.

2. Related work

As an extension of the single mode resource-constrained project scheduling (RCPSP) problem, the MRCPSP is NP-hard [5], with an effective formulations presented by Talbot [44] as follows.

$$\text{Minimize} \sum_{\tau=es(n+1)}^{ls(n+1)} \tau\delta_{n+1,1,\tau} \tag{2.1}$$

$$\sum_{m=1}^{M_i} \sum_{\tau=es(i)}^{ls(i)} \delta_{i,m,\tau} = 1 \qquad 1 \le i \le n \tag{2.2}$$

$$\sum_{m=1}^{M_i} \sum_{\tau=es(i)}^{ls(i)} (\tau + d_i^m)\delta_{i,m,\tau} \le \sum_{m=1}^{M_j} \sum_{es(j)}^{ls(j)} \tau\delta_{j,m,\tau} \qquad \forall\,(i,j)\ \ i \in P_j \tag{2.3}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{i,m,k}^r \sum_{\tau=\max(t-d_i^m,es(i))}^{\min(\tau-1,ls(i))} \delta_{i,m,\tau} \le a_k^r \qquad 1 \le l \le R^r \& 1 \le t \le ls(n+1) \tag{2.4}$$

$$\sum_{i=1}^{n} \sum_{m=1}^{M_i} r_{i,m,l}^e \sum_{\tau=es(i))}^{ls(i)} \delta_{i,m,\tau} \le a_l^e \qquad 1 \le l \le R^e. \tag{2.5}$$

In the above formulation, $\delta_{i,m,\tau}$ is a binary variable which is zero except in the case where activity i has started in mode m at time τ. Also, *es(j)* and *ls(j)* show the earliest and latest start times for activity j and are used to tighten the model. Since the fictitious activity $n+1$ has only one mode, 1, the objective function indicates the starting time of this activity. Relation (2.2) indicates that each activity, with one particular mode, should start only at one instance of time between its earliest start and latest finish times.

Relation (2.3) enforces precedence constraints, in the sense that no activity can start unless all of its predecessors have been completed. Relations (2.4) and (2.5) enforce constraints regarding renewable and non-renewable resources, respectively. As is seen, the formulation does not permit the requirement of in-process activities to exceed the availability of each renewable resource at any point of time. On the other hand the formulation guarantees that the sum of requirements of all activities from each non-renewable resource, which depends on activities' modes, does not exceed the total availability of the corresponding resource.

In [50], solution strategies for the MRCPSP have been classified in eight categories, namely (i) Genetic Algorithms (GA), (ii) Simulated Annealing (SA), (iii) Scatter Search (SS), (iv) Particle Swarm Optimization (PSO), (v) Differential Evolution Algorithms (DEA), (vi) Ant Colony Optimization, (vii) Estimation of Distribution Algorithms (EDA), and (viii) Multi-agent Learning Algorithms (MLA). Also the authors have considered Random Key (RK), and Activity List (AL) as the most widespread schedule representation methods employed. Mode-based *versus* activity-based searches has been discussed in [32].

Moreover, recently, bees algorithms [34], team-based approaches based on different agent cooperation strategies [35], and machine learning heuristics [1] have been developed for solving the MRCPSP. In this section, the related work is classified into 7 subsections and discussed as follows:

2.1. Exact methods including integer and mixed integer programming

In [41], a backtrack-based branch-and-bound type procedure has been presented which performs search-tree reduction towards increasing its performance. First, without considering the resources and only by considering precedence constraints, earliest start and latest start times for each activity is calculated. Then a branch and bound algorithm, where each node of its search-tree represents the execution of a particular activity with a particular mode, implicitly searches all possibilities for different values each node can take.

Mixed-integer linear programming approaches like branch and bound procedures aim at producing exact solutions at the cost of long execution time. The authors in [27] have developed a mixed-integer linear programming approach for solving the MRCPSP. Towards being more efficient, the approach uses Resource-Task-Network (RTN), which is a continuous time-based network representation employed in scheduling process.

In [26] two discrete-time and two continuous-time mathematical models have been presented for the MRCPSP. In both discrete-time models, not only a binary variable has been used to indicate whether or not an activity starts at a particular time but another set of binary variables have been utilized to indicate whether an activity is in process at a particular time.

These double sets of variables, at the cost of increasing the number of binary variables, have simplified the statement of resource constraints and have provided opportunity to tighten the formulation. In the two continuous-time models, on the other hand, both starting and finishing times have been shown with continuous variables. A set of binary variables, $x_{i,j}$, also controls whether the activity i needs to be completed before the activity j starts.

Also the integer programming technique presented in [45] can be considered as one of the effective exact strategies in dealing with MRCPSP. The authors have added local search to their procedure so that it can solve larger problems as well. In effect, incorporating heuristics into exact method has been one of the strategies used by many other researchers to make their exact methods flexible.

A highly effective exact solution strategy has been provided in [43] which has used several efficient solvers to solve the problem. The authors have used a solver independent model to code the problem, so that they can employ different solvers and select the best solution that the solvers produce. Through employing several solvers, the authors have been able to close the sets of benchmark instances with 20 and less activities in the PSPLIB.

2.1.1. Genetic algorithms

In solving optimization problems, genetic algorithms are based on survival of the fittest principle. An effective genetic algorithm to solve the MRCPSP has been employed in [21], which uses both single- and multi-pass improvement to enhance the quality of solutions. A forward-backward improvement technique and a local search have been employed to contribute to the effectiveness of the procedure.

The employed local search uses the concept of multi-mode left shift and tries to left shift activities when possible, systematically enhancing a feasible schedule. Without violating constraints, the multi-mode left shift operation is aimed at reducing the finishing time of an activity without altering either the modes or starting times of other activities.

The fact that the multi-mode left operation never increases the makespan causes the operation to be very beneficial. This operation has first been introduced in [40]. Also another stochastic local search has been employed to find feasible modes with respect to non-renewable resources.

The genetic algorithm presented in [2] employs a two-point forward-backward crossover operator which depending on whether a forward or backward method is supposed to be used, starts to fill the child genome from right or left, respectively. From each two parents, two offspring genomes, a son and a daughter, are generated. The son inherits the forward/backward gene from the father, and the daughter inherits it from the mother.

In their procedure, the mutation is applied to the order of activities as well as to the modes. Through the mutation, not only each activity, with a small chance, is placed in one of its feasible positions in the activity list but the modes are also affected.

One of the other effective genetic algorithms in solving the MRCPSP, is a two-level genetic algorithm presented in [30]. Extending the parameterized schedule generation scheme and using an improvement procedure, the algorithm makes use of two separate levels and in several generations enhances a pool of solutions. Parameterized and polarized adaptive schedule generation schemes [55] are hybrids of serial and parallel schedule generation schemes.

Crossover operators play a key role in the efficiency of genetic algorithms. The innovative crossover operator presented for the MRCPSP in [3] is performed not only on two typical parents but on the best available solution as well. First several cutting points are randomly generated and an offspring genome is partitioned based on these points. Then 1, 2, or 3 is randomly assigned to each partition. Based on the number assigned to each partition, the genes of the corresponding partition are selected from (1) the best available solution, (2) the first parent, and (3) the second parent.

An effective genetic algorithm has been presented in [53] whose efficiency is due to using a powerful population management policy. Based on this policy, several organizations constitute a population. Two splitting and annexing operators adjust the number of such organizations. The set $J30$, which is the set of largest benchmark instances in the PSPLIB [24], has been used to test the performance of the procedure.

A mirror-based genetic algorithm has been provided in [56] which is effective for large number of schedules. Handling both the original and mirrored problems independently, it searches two different solution spaces. Among the two independent solutions found for the original and the mirrored problem, it always selects the best solution. The procedure uses three different crossover operators. These three operators, which are randomly chosen by the same chance are applied on the activity lists, and help each other towards the scanning of two solution spaces the procedure is involved with.

2.1.2. Simulated annealing and local search

Local searches, in general, and simulated annealing algorithms in particular, are based on making local changes to complete solutions. In [6,7], two effective heuristic procedures have been provided for MRCPSP-R. Whereas the first procedure is a single pass technique employing the parallel schedule generation scheme, the second procedure, which works based on the simulated annealing technique, performs local search in the vicinity of high quality solutions and improves them iteratively.

In [25], a procedure has been used which can be identified as one of the effective techniques for tackling the MRCPSP. The procedure starts with a feasible set of modes and maintains the feasibility of the modes in its operations. The procedure has three phases, namely a construction, a local search, and an intensification phase. An initial solution is generated in the construction phase, and the set of feasible modes is searched in the local search, aimed at improving the initial solution. In the intensification phase, the assignment modes proposed in the local search are considered to seek a better solution. The procedure has proved to be very effective.

Also a simulated annealing procedure has been presented in [8] which exploits the characteristics of the MRCPSP. The procedure is based on time incremental process as well as on considering an alternated activity method. In effect, towards neighbourhood exploration, two integrated search loop alternate both modes and activities. For producing a neighbour, an activity is selected randomly and is moved to one of its feasible positions on the activity list.

Activity neighbourhood and mode neighbourhood are performed in two separate stages. First, the order of activities is fixed, and the best modes for that order are sought. Then the best order of activities for the computed modes is searched for. Finding the modes and orders alternatively continues until a halting criterion is researched.

The combination of local search and the forward/backward method is one of the possible approaches in solving the MRCPSP. An algorithm has been presented in [52] which employs both a local search and the

forward/backward method. Founded on a probability based selecting scheme, a feasible tackling procedure uses a multi-mode left shifting rule developed in [40] to improve the completion times of activities.

2.2. Harmony search, scatter search, multi-threaded local search, and path-relinking

Harmony [17] and scatter [19] searches have the common characteristics that use several vector of solutions to produce a new solution. Scatter search uses strategies for search diversification and intensification and its principle is based on formulations dating back to the 1960s in integrating problem constraints and decision rules.

On the other hand, the harmony search is a recent search method which generates random solution vectors store them in harmony memory (HM). This memory not only is used for generating other solution vectors but its worst member regularly is updated with better solutions produced in the search process.

For the MRCPSP, in [49] a scatter search has been developed which uses random keys to encode the solutions, with a serial schedule generation scheme decoding random keys. The authors have extensively analysed the sum of durations, total work content, and mean relative consumption as three different measures.

Since any scatter search needs to use a strategy for search diversification, the authors have used two criteria towards measuring the distance of two solutions. The first criterion is the summation of absolute distances of starting times of same activities, and the second criterion is the number of activities which have used different modes.

In comparison with the scatter search, the harmony search can promote diversification in many different ways. The Harmony Search (HS) presented in [12] has kept effective characteristics of existing metaheursitcs. Similar to Tabu search, it preserves the history of past solutions, and parallel to the simulated annealing search, it varies parameters in the search process. Also, like genetic algorithms, it manipulates a pool of genomes.

Unlike genetic algorithms, which typically select two members of the pool and combine them, the HS creates a new solution by the cooperation of the entire pool. In effect, genetic materials stored in the pool guide the search, and the worst solution in the pool is replaced with any newly generated solution. With a specified chance, which can be changed during the search, a value for a gene is selected among its all possible alleles, rather than among the alleles available in the pool. For each specific gene, all alleles in the pool have the same chance of selection.

In [18], a multi-threaded local search has been provided which operates based on the combination of iterated local and variable neighbourhood searches. The procedure which mainly focuses on the parallel implementation of local searches uses an array of effective local search techniques to improve its performance. In a construction phase, initial solutions are constructed and in an iterative improvement search phase, the initial solutions constructed are improved. The construction phase is composed of two steps, namely the mode assignment and scheduling stages.

An effective path-relinking algorithm has been provided in [33] which operates through connecting two solutions in the search space for creating a path to scan. Through scanning such a path, new solutions are found and the process of connecting two solutions continues. The employed local search is composed of three different moves, namely *OrderSwap*, *Mode1Swap*, and *Mode2Swap*. As the names indicate all of these three moves are based on swap operations, with two of them being related to the change of modes, the other related to altering the order of activities.

2.3. Particle swarm and ant colony optimization

A particle swarm optimization (PSO) technique has been used in [22] to tackle the MRCPSP. With respect to a given measure, PSO is a computational method which tries to iteratively improve a candidature solution, termed as particles, towards optimality. In the search space, each particle moves towards its corresponding local best position as well as the best global position, with all these positions being updated during the search process towards possible improvement.

Unlike PSO which iteratively improves a candidate solution, Ant Colony Optimization (ACO) [14] construct a candidate solution from scratch based on a cooperative learning approach. This is performed based on changing pheromone levels according to information existing in high quality solutions already constructed as well as problem-dependent heuristics. Pheromone levels, which are updated regularly, direct the construction of new solutions. That is why ACO is categorized in constructive methods and not in local searches. An effective ACO has been presented for the MRCPSP in [57].

In this ACO, two problem-dependent heuristics prioritize activities with longer total slacks and shorter execution modes. In this procedure, whereas pheromone evaporation rate contributes to the diversification, updating pheromone levels according to high quality solutions contribute to intensification. The effectiveness of this procedure is partly due to such a proper balance it strikes between diversification and intensification.

2.4. Shuffle Frog Leaping and estimation of distribution algorithms

The Shuffle Frog Leaping [16] and estimation of distribution [29] algorithms are among recently developed population-based metaheursitcs. For the MRCPSP, a Shuffle Frog Leaping algorithm (SFLA) has been presented in [51], and an Estimation of Distribution Algorithm (EDA) has been developed in [38].

The SFLA employs a two-point crossover operator to move a virtual frog towards another. In this SFLA, first, initial agents are positioned in different locations and then partitioned into various subsets called memplexes, each including the same number of elements. Each of these memplexes evolves independently by repeatedly selecting a number of virtual frogs, based on the triangular distribution.

Among the selected virtual frogs, the worst one leaps towards the best virtual frog of the memplex. In the case no improvement has been made, this worst virtual frog leaps towards the best global virtual frog and if it still cannot be improved, it is replaced with a randomly generated virtual frog.

For maintaining diversity, the memplexes are periodically shuffled and new memplexes are formed. Moreover, by performing a local search, the authors have rectified the lack of fine-tuning characteristic of the SFLA. Moreover, before applying the SFLA, the authors have removed the inefficient and non-executable modes.

The EDA unlike the SFLA, constructs solutions based on distribution functions updated based on high quality solutions. The effective method for solving the MRCPSP presented in [38] integrates an EDA with a delete-then-insert local search. Based on the solutions created in the search process, the procedure creates probability distributions guiding the search towards finding high quality solutions.

The activity-mode list, and the multi-mode serial schedule generation scheme have been used for the encoding and decoding purposes, respectively. For the best p solutions generated in each population, the multi-mode double justification is performed to further enhance their quality.

2.5. Hybrids

An evolutionary procedure has been presented in [53], which is very efficient. This efficiency is mainly the result of using an innovative population management, which generates a population composed of several organizations. The number of such organizations is adjustable in the sense that two splitting and annexing operators adjust this number. The procedure has been successfully tested on the set $J30$, which is the largest benchmark instances in the PSPLIB [24].

An effective procedure has been presented in [46] which can be considered as a hybrid of genetic and local searches. The procedure solves problems in two phases, with the first phase being a local search and the second phase further exploring promising areas generated in the first phase. The second phase, which is a genetic algorithm with a large mutation rate, utilizes the solutions generated in the first phase to construct its initial population. In this procedure, the balance between intensification and diversification is maintained in an innovative manner. Similar to the procedure presented in [53], this procedure has also been applied to the set $J30$ successfully.

Satisfiablity Testing (SAT) [10], non-greedy heuristics [11], and automatic algorithm selection [31] are among effective methods applied to the MRCPSP, and MRCPSP/R. Classified as other methods, these three algorithms are briefly discussed here.

Two non-greedy heuristics applied to MRCPSP-R [11] are based on the fact that eliminating unnecessary vacant spaces in the Gant Chart and backward-forward method have proved to be very effective for the RCPSP. In the employed non-greedy modes, an eligible activity which can be executed in a comparatively time-consuming mode, can be put in a waiting status so that a faster mode to become available.

The procedure presented in [31] is based on automatic algorithm selection. This is performed based on the characteristics of the problem instance to be solved. One of these characteristics is the number of precedence relations. The theoretical maximum for the number of these relations is $n(n-1)/2$, where n is the number of activities. The other characteristic is resource strength, introduced in [23].

Resource strength simply shows the relationship between resource availability and resource requirement with considering precedence relations. $M5$ rule and $M5$ tree-model, in data mining, have been used to determine how the characteristics of a problem instance affect the selection of one of the two algorithms controlled by this automatic algorithm selection mechanism.

In [4], a procedure has been presented which combines Mont-Carlo and hyper-heuristic methods. The authors have tested their procedure on some benchmark instances and shown its effectiveness. Some novel neighbourhood moves as well as a memetic algorithm have been employed, which contributes to the effectiveness of the procedure. The procedure has been designed to exploit multicore computing power, and a large majority of its runtime is spent on improving initial solutions.

As another hybrid, a procedure has been provided in [39] for scheduling large-scale multimode projects. As its main components, it uses a novel heuristic and a genetic algorithm. Whereas through dynamic selection of modes, the employed heuristic is involved with efficient resource utilization, the genetic algorithm further improves the quality of the solution generated by the heuristic. A backward-forward mechanism has been employed to further improve the quality of produced solutions.

In [9], an efficient hybrid differential evolution method has been presented which works based on the serial method. The method employs both a fuzzy c-mean clustering technique and a chaotic routine, and incorporates them into a differential evolution procedure. Whereas the role of the fuzzy c-mean clustering technique is to facilitate several multi-parent crossover operators, the chaotic routine is aimed at preventing premature convergence. The termination criterion is met when the generation limit or the maximum number of function evaluation is encountered.

As an another effective hybrid, a hybridization of genetic algorithms and fully informed particle swarm optimization can be mentioned which has been discussed in [36]. In this procedure, a random key has been used as the representation scheme and the serial generation scheme has been used as a decoding mechanism. A pre-processing which reduces the search space has been used to make the search mechanism efficient. Moreover, a fitness function, which penalizes infeasible solutions, guides the search, and a method called multi-mode forward-backward iteration has been used to improve the search efficiency.

3. The Local-search Evolutionary Decomposition-based Procedure (LEDP)

The LEDP relies on the combination of four components, namely (i) a biased-random sampling technique, (ii) a local search, (iii) a decomposition technique, and (iv) an evolutionary search mechanism.

It also employs the method used in [40] as a pre-processing procedure for excluding non-executable and inefficient modes. Figure 3 shows a flow chart of the LDEP.

As is seen, four layers are involved, with each of the first three layer passing their output to their next layer for further improvement. The first layer generates an initial schedule through biased random sampling technique. Mode filtering and prioritizing activities are two operations performed in the first layer.

In the biased random sampling, all modes are set randomly and if this random setting causes non-renewable resource infeasibility, modes are altered randomly towards improving feasibility. If for a particular number of

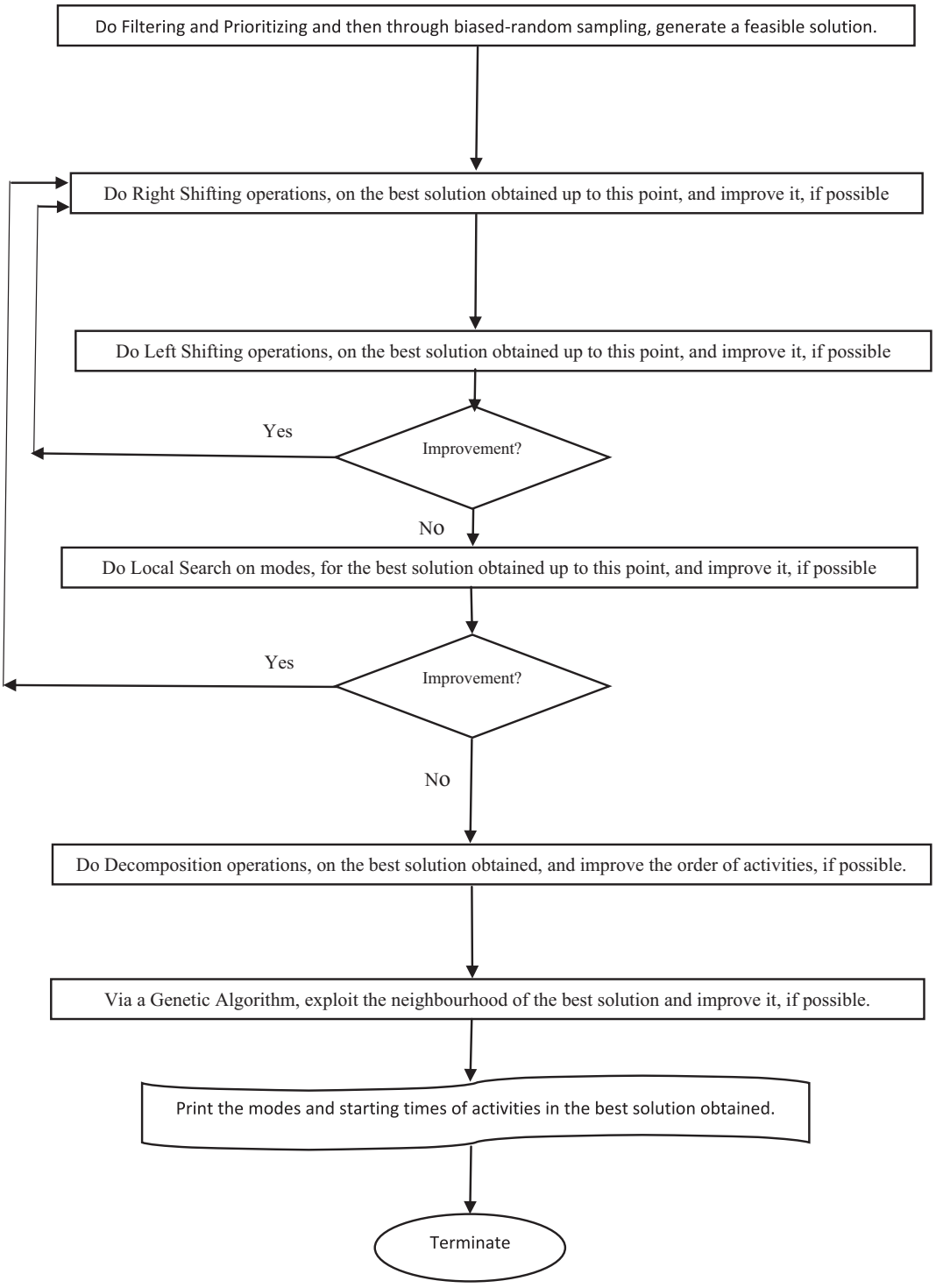


FIGURE 3. A simple flow chart of the LDEP.

times a random change of modes cannot lead to feasibility, the biased random sampling is repeated to alter the initial values of modes.

After setting modes, LFT priority rule [20] is used to create a regret-based biased random activity list. The randomly constructed modes and activity list comprise the initial schedule. Having the initial schedule as its input, the second layer improves its input through a local search. Left- and right-shift operations are the base of the second layer, which produces a local optimal schedule.

As the flow chart shows, right-shifting operations start to change the order of activities and continue as far as these operations can improve the makespan. Then left-shifting operations start and proceed. When neither right- nor left-shifting can make any further improvement, their processes are terminated, and mode modification takes control.

Mode modification starts and as far as the change of mode of any activity, without disturbing feasibility, can improve the makespan, the corresponding mode is changed. As the flow chart shows, even if the mode modification process can improve the mode of a single activity, right- and left-shifting operations resume. A local optimal solution is obtained when right-shifting, left-shifting, and mode modification can no further make any improvement. In other words, the obtained local optimal solution is locally optimal with respect to three neighbourhoods, namely right-shifting, left-shifting, and mode modification.

Receiving the local optimal schedule as its input, the third layer repeatedly decomposes the schedule and tries to fine-tune its different parts. In this layer, modes are untouched and only the starting times are modified. The exact optimization and integration are two operations involved in this layer.

Whereas the exact optimization operation is aimed at finding the optimal schedules of subprojects constructed, the integration operation embeds these optimized sub-schedules into the larger schedule. As the result of these two operations, the third layer is aimed at the further improving of the local optimal schedule.

The modified schedule is used as the input of the fourth layer, which is the genetic algorithm and aims to enhance this modified solution through filling a pool of its slightly perturbed copies. The crossover and mutation are two operations involved in this last layer.

3.1. The pseudocode of the LDEP

Figure 4 shows the pseudocode of the LDEP. At line 5 of the pseudocode, the local search component is called which operates based on the two neighbourhood schemes of right- and left-shift moves. Whereas in an activity list, a right-shift move increases the order of an activity, the left shift move decreases such an order.

Since in a majority of cases, shifting an activity to right or left does not affect the resultant schedule, towards increasing diversification, the employed local search has also been equipped with a swap mechanism. In effect, as well as the modification which is performed as a result of applying a normal local search and causes an activity to randomly change its position in the activity list, two activities in the activity list can also be swapped. The selected activities for being swapped should have no precedence relation.

3.2. The local search

Figure 5 shows the pseudocode of the local search component. As is shown in lines 7, 13, and 26 of the pseudocode, the employed local search works in the two levels of the mode and order of activities.

It first maintains the modes of activities and changes the orders of activities in lines 4–20. Upon reaching stagnation and achieving no further improvement through such alternations, it starts to alter modes in lines 23–35. This continues as far as any improvement can occur. In this way, the modification of modes and order of activities, one after another, is repeated, in lines 3–37, until one can help the other to make further improvement.

A Flag called *Enhancement* controls which loop to be executed. Initially being Set at line 2 of the pseudocode to true, this flag is set to false at line 5 to indicate that only if the right or left shifting can make an improvement, the loop in lines 4–20 can be repeated. This loop changes the order of activities. Hence, if this flag is false at line 20, instated of the loop starting at line 4, the loop starting at line 23 takes control, and different modes for

```
1 PROCEDURE LEDP() // Local-search Evolutionary Decomposition-based Procedure
2       Read the data of the project as input.
3       With using the biased random sampling method, generate an initial schedule for the project.
4       Call the initial schedule as W.
5       Perform a local search on W to possibly improve it.
6       IF (improvement is made)
7               SET W to the improved schedule.
8       ENDIF
9       Perform the decomposition-based search on W.
10      IF (improvement is made)
11              SET W to the improved solution.
12      ENDIF
13      Fill an initial pool with the perturbed copies of W.
14      Perform the genetic algorithm on the pool.
15      IF (improvement is made)
16              SET W to the improved schedule.
17      ENDIF
19      Display W, as the final schedule for the project.
20 ENDPROCEDURE
```

FIGURE 4. The pseudocode of the LEDP.

activities are tested. Line 24 sets the *Enhancement* flag to false to indicate that only if the change of modes can make an improvement, the loop in lines 23–35, which change the modes, can be repeated.

Another flag, called *Restart*, controls whether after changing the modes of activities, another round for changing the order of activities starts or not. As is seen in line 22, before starting to test different modes, the *Restart* flag is set to false, and only if during testing the modes, an improvement is possible this flag is set to true, line 31.

When the *while-loop* making changes in modes, lines 23–35, is terminated, depending of whether or not the value of *Restart* flag is true, the main loop, lines 3–37, restarts or terminated, respectively. Lines 7 and 13 show that two different modules have been called to do right- and left- shifting as well as swapping. The pseudocode of each of these two modules has been presented in Figure 6.

As is seen in both of the pseudocode pieces, which perform reverse operations in comparison to one another, several statements are common, including lines 2–4, 7–9, 11–15 and 17–19.

3.3. The decomposition module

Decomposition has successfully been used in the context of the RCPSP [13,42,54]. In the LEDP, the decomposition technique used in [54] has been modified and upon the termination of the local search, takes control. This module alters the starting times of activities and is aimed at improving the result obtained by the local search. Figure 7 shows the pseudocode of the decomposition module, which aims at enhancing the output of the local search by repeatedly re-scheduling its different sub-schedules to optimality and recombining the results.

For this purpose, it keeps the modes of activities as calculated in the local search, and only modifies the starting times of activities. It decomposes the project into smaller subprojects, based on the solution provided in the local search, and schedules the decomposed subprojects to optimality through its exact optimization part.

```
 1 PROCEDURE  LocalSearch () //performing a local search on both activity and mode lists
 2    SET Enhancement to true. //for starting the two nested while loops
 3    WHILE (Enhancement is True) //the first level loop
 4          WHILE (Enhancement is True) //the  second level loop
 5               SET Enhancement to False. //it will change to True if any improvement occurs.
 6               FOR i=1 to NumberofActivities // checking the positions on the activity list
 7                    Call  ShiftRightAndSwap( i )
 8                    IF calling ShiftRightAndSwap has caused any enhancement to the solution
 9                         SET Solution to the improved solution.
10                         SET Enhancement to True.
11                         COUNTINUE FOR  // increase i by 1 and continue the FOR loop from start
12                    ENDIF
13                    Call  ShiftLeftAndSwap( i )
14                    IF calling ShiftLeft has caused any enhancement to the solution
15                         SET Solution to the improved solution.
16                         SET Enhancement to True.
17                         COUNTINUE FOR  // increase i by 1 and continue the for loop from start
18                    ENDIF
19               ENDFOR
20          ENDWHIILE
21          SET Enhancement to true. //for the while loop to start
22          SET Restart to False. //It will be set to True if any improvement occurs
23          WHILE (Enhancement is True)
24               SET Enhancement to False. //It will be set to True if any improvement occurs
25               FOR each activity j
26                    FOR each m as a feasible mode different from the current mode of activity j
27                         Test m as a new mode of activity j.
28                         IF( the test does not violate feasibility and has resulted in some enhancement)
29                              Change the mode of activity j to m.
30                              SET Enhancement to True.
31                              SET Restart to True.
32                         ENDIF
33                    ENDFOR
34               ENDFOR
35          ENDWHILE
36          SET Enhancement to Restart //for whether or not to start the main while loop
37     ENDWHIILE
38 ENDPROCEDURE
```

FIGURE 5. The pseudocode of the employed local search.

The pseudocode starts with line 2, which sets *interval* to the makespan calculated in the local search. Initially, a range with the size of *interval* is considered, and all activities in this range comprise a subproject, with this subproject being tried to be solved to optimality. If within a short period of time (several milliseconds), this subproject cannot be solved to optimality, the subproject is further decomposed to smaller subprojects, and this

```
 1 PROCEDURE ShiftRightAndSwap(i) //shifting the i-th activity on the activity list to right and swapping.
 2       SET OriginalActivityList to the current activity list //keeping the current activity list
 3       SET ImprovedActivityList to the current activity list //Default value for ImprovedActivityList
 4       SET j o i. //j+1 points to the index of location located on the right side of location i on the activity list.
 5       WHILE (  (j - i) <Radius and j < NumberOfActivities)
 6               IF (the activity located in position j+1 is the successor of that in location j)
 7                   SET the activity list to OriginalActivityList.//cancelling all unfruitful changes
 8                   BREAK WHILE
 9               ENDIF
10               Shift the activity located in position j of activity list to the position j+1.
11               Swap two non-predecessor-successor neighboring activities in the activity list randomly.
12               IF (shifting along with swapping provides an improvement over OriginalActivityList)
13                   SET ImprovedActivityList to the new activity list.
14                   BREAK WHILE
15               ENDIF
16               Add 1 to j.
17       ENDWHILE
18       RETURN ImprovedActivityList
19  ENDPROCEDURE
```

```
01 PROCEDURE ShiftLeftAndSwap(i) //shifting the i-th activity on the activity list to left and swapping.
02       SET OriginalActivityList to the current activity list //keeping the current activity list
03       SET ImprovedActivityList to the current activity list //Default value for ImprovedActivityList
04       SET j o i. //j-1 points to the index of locations located on the right side of location i on the activity list.
05       WHILE (  (i - j) <Radius and j  > 1)
06               IF (the activity located in position j-1 is the predecessor of that in location j)
07                   SET the activity list to OriginalActivityList.//cancelling all unfruitful changes
08                   BREAK WHILE
09               ENDIF
10               Shift the activity located in position j of activity list to the position j-1.
11               Swap two non-predecessor-successor neighboring activities in the activity list randomly.
12               IF (shifting along with swapping provides an improvement over OriginalActivityList)
13                    SET ImprovedActivityList to the new activity list.
14                    BREAK WHILE
15               ENDIF
16               Decrease 1 from j.
17       ENDWHILE
18       RETURN ImprovedActivityList
19 ENDPROCEDURE
```

FIGURE 6. Two pieces of pseudocode for right- and left-shifting and swapping.

```
 1 PROCEDURE Decompose() //It updates the original schedule by repeatedly scheduling different parts of project.
 2     SET Interval to the current makespan of the project. //Interval shows the length of the subproject to be optimized.
 3     SET IsOpimalFound to False//a default value indicating no optimal schedule for the entire project is available yet.
 4     SET IsRepeatNeeded to True.//default value for starting the main while loop.
 5     WHILE (IsRepeatNeeded is True) //start to create subprojects form from the start of the project.
 6         SET IsRepeatNeeded to False. //it will be set to True whenever any improvement is made in the process.
 7         SET StartingPoint to 0. //StartingPoint shows the starting point of the project from which a subproject is built.
 8         WHILE (TimeLimit is not reached)
 9             IF (StartingPoint is outside the range of the project) BREAK WHILE
10             SET EndingPoint to StartingPoint plus Interval.
11             Select the part of project between StartingPoint and EndingPoint and call it SubProject.
12             Try to schedule the SubProject to optimality within the given milliseconds and with fixed modes.
13             IF (the optimal schedule is obtained) //the subproject  with fixed modes has been solved to optimality.
14               IF (StartingPoint is 0 and Interval equals the length of the project)
15                 SET IsOpimalFound to True.
16                 BREAK WHILE //no extra computational effort is needed because the optimal schedule is obtained.
17               ELSE //because the subproject has been solved to optimality, make Interval larger for the next round.
18                 Embed the optimized sub-schedule obtained into the original schedule.
19                 IF (embedding causes any improvement in the original schedule)
20                     SET IsRepeatNeeded to True. //signifying that decomposition has been beneficial
21                 ENDIF //since optimal schedule obtained, both StartingPoint and Interval are increased.
22                 IF (EndingPoint shows the end of the project) BREAK WHILE
23                 Multiply Interval By IntervalExpansionRatio //IntervalExpansionRatio > 1
24                 Multiply StartingPoint by StartingPointExpansionRatio // StartingPointExpansionRatio >1
25               ENDIF
26             ELSE //the subproject is too large for its optimal schedule to be obtained.
27                 Multiply Interval By IntervalShrinkingRatio //IntervalShrinkingRatio is smaller than 1
28           ENDIF
29         ENDWHILE
30     ENDWHILE
31 ENDPROCEDURE
```

FIGURE 7. The pseudocode of the decomposition search.

process continues until they can be solved to optimality. Upon solving a subproject to optimality, its optimal schedule is embedded in the original schedule in the hope of decreasing the overall makespan.

The constructing of a subproject starts with line 7 which initially sets its starting time to zero. Later, this starting time is modified in line 24. Line 11 selects the part of project between *StartingPoint* and *EndingPoint* and calls it *SubProject*. In the current schedule, all activities which have been located in this range become part of this constructed subproject. Upon constructing the subproject, line 12 tries to schedule it to optimality within a specified time-limit, say several milliseconds, with current modes.

If the optimal schedule can be obtained within the time-limit, lines 14–25 take control, and otherwise line 26 makes the interval smaller, hoping that the smaller subproject is solved to optimality. As is seen in the pseudocode, the process of making a subproject smaller continues until the subproject can be solved to optimality. In the case where the subproject is solved to optimality, lines 14–25 take control. Lines 14–16 check whether the optimal schedule found is for the entire project and in this case, the module is terminated.

Any optimized sub-schedule is embedded into the original schedule at line 18. Line 19 checks whether embedding the sub-schedule into the original schedule has caused any improvement. If this embedding has led to any

improvement, line 20 sets *IsRepeatNeeded* to true, signifying that the decomposition has been beneficial and can be repeated.

As is seen, the value of *IsRepeatNeeded* determines whether or not line 5 should start another round of decomposition, with the new round of decomposition operating on the improved schedule, and aimed at further enhancing of the schedule. Also whenever a subproject is solved to optimality, *Interval* becomes larger for the next round, in the hope that the larger subproject can be solved to optimality. The decomposition component is terminated whenever for a round, which includes lines 5-30, the value of *IsRepeatNeeded* remains false.

3.4. Searching the vicinity of the best solution obtained

Searching the vicinity of best solutions through genetic algorithms is a successful strategy [47]. That is why when the decomposition component is terminated, the LDEP searches the vicinity of the best solution obtained. The employed genetic algorithm used for this purpose is similar to the one presented in [56], with the two main differences. The first difference is that it does not use mirroring operations, and the second difference is that instead of using three crossover operators on the activity list, it uses a single crossover operator.

The employed genetic algorithm starts with slightly perturbing the copies of the solution constructed and then using both crossover and mutation operations to improve the content of the pool in several generations.

Each genome in the pool consists of two separate partitions. Whereas the first partition includes the modes of activities, the second partition contains the orders of activities, as an activity list. Towards fine-tuning the best available solution, crossover and mutation operators alter both modes and orders of activities.

A two-point crossover operator is applied to the activity lists of every two parent genomes and creates the activity list of the corresponding offspring genome. On the other hand, for creating the modes, a uniform crossover operator is applied to the modes and creates a feasible offspring genome.

The feasibility of modes with respect to non-renewable resources is obtained through first all modes being copied from one parent, and then the modes from the second parent being chosen randomly and with the chance of 50 percent replacing the original mode. The same as in [56], this replacement is allowed only if the feasibility is not violated, and both parallel and serial schedules have been used in decoding followed by double justification [48].

4. Computational experiments

The LDEP has been coded in C++, and a PC with 2.5 GHz speed and 8 GB of RAM has been used to test the procedure's performance on 552 benchmark instances. These instances have been taken from the library of project scheduling problems, PSPLIB [24]. Comprising various sets of multi-mode benchmark instances, this library includes the best available solution in the literature for each instance as well. With respect to the latest solutions, the library has been updated in July 2015.

Among the contents of the library, only those instances have been selected which have maximum number of activities, 30, and have feasible solutions, as the projects in the library have between 10–30 activities and some do not have any feasible solution.

4.1. Setting the parameters of the procedure

For setting the parameters of the LDEP, first a small representative subset of these 552 instances was selected, and then the procedure was tested on this subset with varying parameters. Based on the tests performed, swapping rate for the local search was set to 0.00. In effect, since swapping rate incorporates more randomness in the ordering of activities and changes the balance towards exploration, its increase disturbed this balance. The other parameter increasing the weight of exploration against exploitation is mutation rate. After several trials and finding its best performance, this parameter was set to 0.03.

Moreover, with respect to the size of the pool and the number of generations, several tests were performed and these two values were set to 500, and 750, respectively. The experiments on these representative instances showed that for the cases where a limit is set on the number of schedules, the smaller limits require the smaller

pool-sizes. That is why for the limits of 5000, 10 000, 20 000, and 50 000 number of schedules, the population size was set to 30, 50, 80, and 120, respectively.

One of the other parameters that was determined in these experiments, was the time-limit allocated to solving each decomposed problem. Experiments showed that allocating large values to this parameter increased solution time without significantly affecting solution quality. Hence, this parameter was set to the small value of 0.5 second. In the cases where a limit is set on the number of schedules, this small value reduces to zero.

4.2. The performance of the procedure

Based on the above setting, each of 552 employed benchmark instances were solved within the maximum of 17 seconds. The results are promising in the sense that among 552 solutions produced, 512 solutions are among the best available solutions in the literature, comprising over 92.8% of instance, 512/552.

It is conjectured that in a parallel processing environments, if the last component of the procedure is multi-threaded and run on different processors, without any increase in solution time, the procedure can produce even better results. The reason is that the fourth layer can work on several processors simultaneously and fine-tune the solution constructed in its first three layers.

A justification why for these instances, the procedure could not find improved solutions is that some of these best solutions are optimal and cannot be improved at all. Moreover, those instances whose optimal solutions have not yet been found have potentially a very large solutions space. It seems that multi-threaded procedures which can be run on a large number of different cores have better chance for searching such large solution spaces.

To remove the effect of the computers on the performance of procedures, authors usually run their procedures based on different limits set on the number of schedules generated. In other words, rather than limiting the time allocated to their procedures, they limit the number of schedules the corresponding procedure can produce. The reason is that when the number of schedules, rather than allocated time, limits a procedure, no more the speed of the computer on which the procedure is run can affect its output.

In order to compare the procedure with other procedures, we have done the same by removing its third layer, which cannot be controlled based on the number of schedules, and running the remaining procedure for the limits of 5000, 10 000, 20 000, and 50 000 schedules.

For the comparison purposes, only procedures have been considered which have been tested on the set $J30$, which includes the largest set of benchmark instances in the PSPLIB for the MRCSP.

Table 1 shows the results, indicating that for the limits of 10 000 and 20 000, the procedure has obtained better results than those reported for the other procedures. As is seen in the table, the average percentage division from the CPM-lower bounds criterion has been used to measure the performance of the procedures, with smaller deviations signifying better results. The reason for using this criterion is that the optimal solutions

TABLE 1. The performance of the procedure in comparison with those of other procedures based on the limits of 5000, 10 000, 20 000, and 50 000 schedules with the criterion of the average deviation from CPM lower-bounds.

	Number of Schedules Generated			
Reference	5000	10 000	20 000	50 000
This Paper	15.325	13.963	13.171	12.803
Zamani [56]	19.553	14.516	13.312	12.906
Mutritiba *et al.* [33]	12.850	–	–	12.550
Asta *et al.* [4]	–	–	–	13.68
Wang and Liu [53]	17.159	16.029	15.165	14.423
Van Peteghem and Vanhoucke [49]	13.66	–	–	12.720
Tseng and Chen [46]	18.332	16.786	16.193	15.683

for a large majority of these 552 benchmark instances are unknown, and all of the authors have used this alternative criterion to measure the performance of their procedures.

We also calculated the average percentage division from the best available solutions based on the latest solutions in the PSPLIB library, updated in July 2015. It is worth noting that the average percentage division from the best available solutions criterion is not usually used by the authors, as the best solutions change over time. However, when used along with the other criterion, it seems to be an informing piece of information. For instance, when this average percentage is 2, it can be expected that if the best available solution is 100, the solution obtained by the LDEP is expected to be 102.

For the limits of 5000, 10 000, 20 000, and 50 000 schedules, we calculated these percentages and obtained 2.489, 1.327, 0.712, and 0.412, respectively. As is seen, the same as the percentage deviation from the CPM lower-bounds, these percentages becomes smaller as the limit on the number of schedules increases.

Since the value of 33.375 shows the average of the best available solutions reported in the PSPLIB, the performance of the LDEP for the limit of 50 000 schedules can also be stated as follows. For 5 out of 6 cases, the LDEP is expected to find the best available solution and for the remaining case to miss the best available solution only by one unit as $1/(6*33.375) > 0.412\%$. In other words, the value of 0.412, obtained for 50 000 schedules, indicates that for every one unit miss in value, five cases are expected to match with the best available solutions.

4.3. The effect of each layer on the performance of the procedure

With the first layer generating an initial solution, each of the other three layers potentially contributes towards increasing the quality of its previous layer. Solutions never degrade because, in the worst case scenario, each of these layers keeps the solution of its previous layer unchanged. In finding the effect of each of the last three layers on the performance of the procedure, we have used a criterion called the number of improvements.

This criterion is defined as the number of cases in which the corresponding layer can improve the solution of its previous layer. In measuring this criterion, the parameters have been set to the same values discussed in the parameter setting subsection. The results are as follows.

Whereas in 525, out of 552, cases the second layer improves the performance of the first layer, the third layer can improve the performance of the second layer only in 73 out of 552 cases. With this criterion, the fourth layer performs nearly the same as the second layer as it improves the performance of the third layer in 537 cases. The result of these computational experiments justifies the very short computational time that in the parameter setting was allocated to the third layer. In effect, as discussed, in comparing the performance of the procedure with other procedures, this short time was reduced to 0.

In order to further justify the shortness of the time we allocated to the third layer, we increased it 20-fold, from 500 to 10000 milliseconds, and ran the procedure on all 552 instances. The result is that the third layer in this case could only insignificantly improve its performance from 73 to 74.

5. Conclusion

The computational results demonstrate the efficiency of the procedure, and the analysis related to the impact of the four layers of the procedure indicates that two key factors contribute to this efficiency.

First, the simultaneous use two right- and left-shift moves, along with mode moves, makes the employed local search effective, as the solution obtained is local optimal with respect to three different neighbourhoods of right-shift, left-shift and mode moves. That is why in computational experiments, the local search was able to improve the initial solution in 525 out of 552 cases.

Second, the evolutionary search technique, which performs fine-tuning on the best solution obtained by the cooperation of the first three components, adds to the effectiveness of the procedure. In effect, computational experiments revealed that the evolutionary search technique was able to enhance the solution obtained by the local search in 537 out of 552 cases. Hence, filling the initial pool with slightly perturbed copies of the best available solution, and concentrating the effort on the fine-tuning of such a high quality solution was fruitful.

With respect to the shortcomings revealed in computational experiments, the LDEP can be improved in two different directions. First, the decomposition technique, which keeps the modes and alters the starting times of activities, needs to change from an exact method to a heuristic. The reason is that computational experiments revealed that it could improve the solution of its previous stage in only 73 out of 552 cases. By trading-off optimality with flexibility and using a heuristic instead of an exact method, this component can concentrate on improving both the mode and starting time of activities within a reasonable time.

Second, since the computational experiments showed that the local search could consistently improve the initial solution, a better method for the generating an initial solution can be helpful. After all, the local search is a relatively time-consuming task, and it is better to minimize its use through generating better initial solutions.

The computational experiments with respect to the effect of each component clearly showed that the procedure is a multifaceted artefact and its performance not only depends on how each of its different components works but on how much time is allocated to each component. Devising an automatic feedback-based mechanism that depending on the characteristics of the problem at hand allocates a proper time to each component is expected to affect the efficiency of the LDEP, and is of paramount importance.

References

[1] P.I. Adamu, M.C. Agarana and H.I. Okagbue, Machine Learning Heuristic for Solving Multi-Mode Resource-Constrained Project Scheduling Problems (2018).

[2] J. Alcaraz, C. Maroto and R. Ruiz, Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *J. Oper. Res. Soc.* **54** (2003) 614–626.

[3] A. Andreica and C. Chira, Best-order crossover in an evolutionary approach to multi-mode resource-constrained project scheduling. *Int. J. Comput. Inf. Syst. Ind. Manage. (IJCISIM)* **6** (2014) 364–372.

[4] S. Asta, D. Karapetyan, A. Kheiri, E. Özcan and A.J. Parkes, Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. *Inf. Sci.* **373** (2016) 476–498.

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

[6] F.F. Boctor, Heuristics for scheduling projects with resource restrictions and several resource-duration modes. *Int. J. Prod. Res.* **31** (1993) 2547–2558.

[7] F.F. Boctor, Resource-constrained project scheduling by simulated annealing. *Int. J. Prod. Res.* **34** (1996) 2335–2351.

[8] K. Bouleimen and 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** (2003) 268–281.

[9] M.-Y. Cheng and D.-H. Tran, An efficient hybrid differential evolution based serial method for multimode resource-constrained project scheduling. *KSCE J. Civil Eng.* **20** (2016) 90–100.

[10] J. Coelho and M. Vanhoucke, Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *Eur. J. Oper. Res.* **213** (2011) 73–82.

[11] S. Colak, A. Agarwal and S. Erenguc, Multi-mode resource-constrained project-scheduling problem with renewable resources: new solution approaches. *J. Bus. Econ. Res. (JBER)* **11** (2013) 455–468.

[12] A. Csébfalvi and E. Szendrői, An improved hybrid method for the multi-mode resource-constrained project scheduling problem. In: *Proceedings of the Eighth International Conference on Engineering Computational Technology*. Civil-Comp Press, Stirling, UK (2012).

[13] D. Debels and M. Vanhoucke, A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. *Oper. Res.* **55** (2007) 457–469.

[14] M. Dorigo and L.M. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1** (1997) 53–66.

[15] S.E. Elmaghraby, Activity networks: project planning and control by network models. New York Wiley (1977).

[16] M. Eusuff, K. Lansey and F. Pasha, Shuffled frog-leaping algorithm: a memetic meta-heuristic for discrete optimization. *Eng. Optim.* **38** (2006) 129–154.

[17] Z.W. Geem, J.H. Kim and G. Loganathan, A new heuristic optimization algorithm: harmony search. *Simulation* **76** (2001) 60–68.

[18] M.J. Geiger, A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem. *Eur. J. Oper. Res.* **256** (2017) 729–741.

[19] F. Glover, J.P. Kelly and M. Laguna, Genetic algorithms and tabu search: hybrids for optimization. *Comput. Oper. Res.* **22** (1995) 111–134.

[20] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling. *Nav. Res. Logist.* **45** (1998) 733–750

[21] S. Hartmann, Project scheduling with multiple modes: a genetic algorithm. *Ann. Oper. Res.* **102** (2001) 111–135.

[22] B. Jarboui, N. Damak, P. Siarry and A. Rebai, A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. *Appl. Math. Comput.* **195** (2008) 299–308.

[23] R. Kolisch, A. Sprecher and A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems. *Manage. Sci.* **41** (1995) 1693–1702.
[24] R. Kolisch and A. Sprecher, PSPLIB – a project scheduling library. *Eur. J. Oper. Res.* **96** (1996) 205–216.
[25] R. Kolisch and A. Drexl, Local search for nonpreemptive multi-mode resource-constrained project scheduling. *IIE Trans.* **29** (1997) 987–999.
[26] G.M. Kopanos, T.S. Kyriakidis and M.C. Georgiadis, New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. *Comput. Chem. Eng.* **68** (2014) 96–106.
[27] T.S. Kyriakidis, G.M. Kopanos and M.C. Georgiadis, MILP formulations for single-and multi-mode resource-constrained project scheduling problems. *Comput. Chem. Eng.* **36** (2012) 369–385.
[28] E.L. Lawler, J.K. Lenstra and A.H.G.R. Kan, Recent developments in deterministic sequencing and scheduling: a survey. In: Deterministic Stochastic Scheduling. Springer, Dordrecht (1982) 35–73.
[29] J.A. Lozano, R. Sagarna and P. Larrañaga, Parallel estimation of distribution algorithms. In: Estimation of Distribution Algorithms. Springer (2002) 129–145.
[30] J. Magalhães-Mendes, A two-level genetic algorithm for the multi-mode resource-constrained project scheduling problem. *Int. J. Syst. Appl. Eng. Dev.* **5** (2011) 271–278.
[31] T. Messelis and P. De Causmaecker, An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **233** (2014) 511–528.
[32] D. Morillo, F. Barber and M.A. Salido, Mode-based *versus* activity-based search for a nonredundant resolution of the multimode resource-constrained project scheduling problem. *Math. Prob. Eng.* **2017** (2017).
[33] A.E.F. Muritiba, C.D. Rodrigues and F.A. da Costa, A path-relinking algorithm for the multi-mode resource-constrained project scheduling problem. *Comput. Oper. Res.* **92** (2018) 145–154.
[34] E. Oztemel and A.A. Selam, Bees algorithm for multi-mode, resource-constrained project scheduling in molding industry. *Comput. Ind. Eng.* **112** (2017) 187–196.
[35] E. Ratajczak-Ropel, Experimental evaluation of agent-based approaches to solving multi-mode resource-constrained project scheduling problem. *Cybern. Syst.* (2018) 1–21.
[36] M. Sebt, M. Afshar and Y. Alipouri, Hybridization of genetic algorithm and fully informed particle swarm for solving the multi-mode resource-constrained project scheduling problem. *Eng. Optim.* **49** (2017) 513–530.
[37] R. Slowinski, Two approaches to problems of resource allocation among project activities – a comparative study. *J. Oper. Res. Soc.* **31** (1980) 711–723.
[38] O.S. Soliman and E.A. Elgendi, A hybrid estimation of distribution algorithm with random walk local search for multi-mode resource-constrained project scheduling problems. *Int. J. Comput. Trends Tech (IJCTT)* **8** (2014) 57–64.
[39] R. Sonmez and M. Gürel, Hybrid optimization method for large-scale multimode resource-constrained project scheduling problem. *J. Manage. Eng.* **32** (2016) 04016020.
[40] A. Sprecher, S. Hartmann and A. Drexl, An exact algorithm for project scheduling with multiple modes. *OR Spectr.* **19** (1997) 195–203.
[41] A. Sprecher and A. Drexl, Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *Eur. J. Oper. Res.* **107** (1998) 431–450.
[42] A. Sprecher, Network decomposition techniques for resource-constrained project scheduling. *Oper. Res. Soc.* **53** (2002) 405–414.
[43] R. Szeredi and A. Schutt, Modelling and solving multi-mode resource-constrained project scheduling. In: *International Conference on Principles and Practice of Constraint Programming*. Springer (2016).
[44] F.B. Talbot, Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. *Manage. Sci.* **28** (1982) 1197–1210.
[45] T.A. Toffolo, H.G. Santos, M.A. Carvalho and J.A. Soares, An integer programming approach to the multimode resource-constrained multiproject scheduling problem. *J. Schedul.* **19** (2016) 295–307.
[46] L.-Y. Tseng and S.-C. Chen, Two-phase genetic local search algorithm for the multimode resource-constrained project scheduling problem. *IEEE Trans. Evol. Comput.* **13** (2009) 848–857.
[47] V. Valls, F. Ballestin and S. Quintanilla, A population-based approach to the resource-constrained project scheduling problem. *Ann. Oper. Res.* **131** (2004) 305–324.
[48] V. Valls, F. Ballestin and S. Quintanilla, Justification and RCPSP: a technique that pays. *Eur. J. Oper. Res.* **165** (2005) 375–386.
[49] V. Van Peteghem and M. Vanhoucke, Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *J. Heuristics* **17** (2011) 705–728.
[50] V. Van Peteghem and M. Vanhoucke, An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *Eur. J. Oper. Res.* **235** (2014) 62–72.
[51] L. Wang and C. Fang, An effective shuffled frog-leaping algorithm for multi-mode resource-constrained project scheduling problem. *Inf. Sci.* **181** (2011) 4804–4822.
[52] L. Wang and C. Fang, An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem. *Comput. Oper. Res.* **39** (2012) 449–460.
[53] L. Wang and J. Liu, Solving multimode resource-constrained project scheduling problems using an organizational evolutionary algorithm. In: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems*. Springer (2015).
[54] R. Zamani, A hybrid decomposition procedure for scheduling projects under multiple resource constraints. *Oper. Res.* **11** (2011) 93–111.

[55] R. Zamani, A polarized adaptive schedule generation scheme for the resource-constrained project scheduling problem. *RAIRO: OR* **46** (2012) 23–39.
[56] R. Zamani, An effective mirror-based genetic algorithm for scheduling multi-mode resource constrained projects. *Comput. Ind. Eng.* **127** (2019) 914–924.
[57] H. Zhang, Ant colony optimization for multimode resource-constrained project scheduling. *J. Manage. Eng.* **28** (2012) 150–159.