www.palgrave-journals.com/jors/

A path-relinking metaheuristic for the resource levelling problem

M Ranjbar*

Ferdowsi University of Mashhad, Iran

This paper deals with project scheduling problem with resource levelling objective function where precedence relations among activities are prescribed. We develop a dedicated path-relinking metaheuristic algorithm to tackle this problem. Computational results on randomly generated test sets indicate the developed procedure is efficient and outperforms the best available metaheuristic algorithms in the literature.

Journal of the Operational Research Society (2013) **64,** 1071–1078. doi:10.1057/jors.2012.119
Published online 3 October 2012

Keywords: project scheduling; resource levelling problem; path-relinking

1. Introduction

Since fluctuation cost in the resources utilization of a project is high, scheduling of a project has to be modified such that variation in profile of all resources is minimized. This issue is addressed to the *resource levelling problem* (RLP) in the literature of project management. The RLP involves the non-preemptive scheduling of project activities subject to precedence constraints in order to minimize the variation of resources' requirements (such as equipment, manpower, etc). As it has been shown (Tavares, 1987), the RLP plays a significant role in practice because large variations in resources utilization can be very costly. Also, some resources may not be available in the quantities management needs. There is a variety of applications for the RLP. The RLP has been applied in practice by developing software tools like Primavera and SuperProject in which simple shifting heuristic procedures are applied (Meridith and Mantel, 1995). In addition to the project management, the resource levelling has been used in make-to-order production (Ballestin *et al*, 2007) and in machine scheduling environment (Drotos and Kis, 2011).

In terms of resource constraints, there are two types of RLP in practice, that is, with resource constraints and without resource constraints. Both types are considered in the literature of project management but the RLP without resource constraints is the subject of this research. In addition, there are two types of RLP in terms of precedence constraints, that is, special minimum time lags (temporal precedence relations) and general minimum and maximum time lags (general precedence relations). In this paper, we assume there are temporal precedence relations among activities.

As has been shown (Neumann *et al*, 2003), the RLP is NP-hard in ordinary sense even if one resource is considered. Several exact, heuristic and metaheuristic procedures have been developed for the RLP. For the RLP with temporal precedence constraints, exact algorithms based on implicit enumeration, integer programming, or dynamic programming techniques have been created (Ahuja, 1976; Bandelloni *et al*, 1994; Demeulemeester, 1995; Younis and Saad, 1996 and Easa, 1998). One of the first heuristics for the RLP is due to Burgess and Killebrew (Burgess and Killebrew, 1962). This procedure is applicable to CPM/PERT networks consisting of activities and temporal relations between them. It aims at finding the best start times of the activities by shifting them to the right step by step in several rounds. For this problem, other heuristic procedures have been proposed (Ahuja, 1976; Harris, 1978 and 1990; Moder *et al*, 1983 and Takamoto *et al*, 1995) where most of them represent simple shifting heuristics or priority rule methods. Also, a neural network approach for solving the RLP has been developed (Savin *et al*, 1996 and 1997). Furthermore, metaheuristic procedures based on genetic algorithm, multi-objective genetic algorithm and ant colony algorithm have been devised (Leu *et al*, 2000; Roca *et al*, 2008 and Geng *et al*, 2011).

For the RLP with general precedence relations, two researches (Brinkmann and Neumann, 1996 and Neumann and Zimmermann, 1999) have devised priority-rule based heuristics, where the latter represents the first heuristics for resource levelling with resource constraints in the open literature. Also, a local search heuristic and a branch-and-bound algorithm are proposed (Neumann and Zimmermann

Correspondence: M Ranjbar, Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, P.O. Box 91775-1111, Iran.
E-mail: m_ranjbar@um.ac.ir

2000). The computational results of the branch-and-bound method are limited to instances with 20 tasks only. For a recent review of resource constrained project scheduling, see Hartmann and Briskorn (2010). More information on the topic, including several models and algorithms, can be found in Demeulemeester and Herroelen (2002).

In this paper, a path-relinking (PR) metaheuristic algorithm including relinking method, repairing method, and improvement method is developed to tackle the RLP instances. The remainder of this paper is organized as follows. The problem description and modelling is presented in Section 2. In Section 3, we provide the sketch of the solution approach. Computational results are discussed in Section 4. Finally, a summary and outlook on further research are given in Section 5.

2. Problem description and modelling

Suppose that set $N=\{0,1,\ldots,n+1\}$ represents activities of a project in which dummy activities 0 and $n+1$ indicate start and end of the project. We use the activity-on-node (AON) representation to show the project graphically in which each node indicates an individual activity and each directed arc (i,j) shows a finish-to-start type precedence relation between activities $i,j\in N$. Also, assume set $A=\{(i,j)|i,j\in N\}$ represents the temporal precedence relations among activities and $R=\{1,\ldots,m\}$ shows the set of renewable resources. Each activity $i\in N$ has a fixed duration d_i and a constant resource requirement r_{ik} to the renewable resource k where d_i and r_{ik} are both non-negative integers. Each solution (schedule) is represented by a vector of start times $\mathbf{S}=(s_0,s_1,\ldots,s_{n+1})$ where s_i indicates the start time of activity $i\in N$. If we establish the convention that $s_0=0$, which means that the project always begins at time zero, s_{n+1} coincides with the project duration. In addition, a deadline δ is given for the project completion. The RLP can be modelled as follows:

$$\operatorname{Min} Z=f(\mathbf{S}) \tag{1}$$

s.t.

$$s_j-s_i\geqslant d_i \quad \forall(i,j)\in A \tag{2}$$

$$s_i\in\mathbb{Z}^+ \quad \forall i\in N \tag{3}$$

$$s_0=0 \tag{4}$$

$$s_{n+1}\leqslant\delta \tag{5}$$

The objective function $f(\mathbf{S})$ is related to smoothing of the resource utilization and there are several different formulas for that in the literature (Neumann and Zimmermann, 2000). The most commonly used formula is $f(\mathbf{S})=\sum_{k=1}^{m}c_k\sum_{t=1}^{s_{n+1}}(r_k(\mathbf{S},t)-r_k(\mathbf{S},t-1))^2$, where $r_k(\mathbf{S},t)$ shows the resource utilization of resource k over time periods t (interval $[t-1,t]$) where $t=1,\ldots,s_{n+1}$ and $c_k>0$ represents the cost per change in resource utilization of resource $k\in R$. In other words, $r_k(\mathbf{S},t)$ can be represented as $r_k(\mathbf{S},t)=\sum_{i\in B_k(\mathbf{S},t)}r_{ik}$, where $B_k(\mathbf{S},t)$ indicates a set of activities that are in progress on resource k and at time period t in schedule $\mathbf{S}$. For initialization, we assume for each resource k and schedule $\mathbf{S}$, $r_k(\mathbf{S},0)=0$. It has been proved that minimization of $\sum_{k=1}^{m}c_k\sum_{t=1}^{s_{n+1}}(r_k(\mathbf{S},t)-r_k(\mathbf{S},t-1))^2$ corresponds to minimization of $\sum_{k=1}^{m}c_k\sum_{t=1}^{s_{n+1}}r_k(\mathbf{S},t)^2$ (Burgess and Killebrew, 1962). Thus, after this, for convenience we consider $f(\mathbf{S})=\sum_{k=1}^{m}c_k\sum_{t=1}^{s_{n+1}}r_k(\mathbf{S},t)^2$. Constraint (2) indicates the temporal precedence constraints while constraint (3) is related to the non-negative values of start times ($\mathbb{Z}^+$ indicates the set of non-negative integers). As we mentioned before, we assume the project is started at time zero and this assumption is shown by constraint (4). Finally, constraint (5) implies all activities must be finished before or at the given deadline δ.

3. The path-relinking metaheuristic for the RLP

3.1. The optimal solution property

Our developed metaheuristic is constructed based on an optimal solution property of the RLP developed in Theorem 3 of Neumann and Zimmermann (1999). Assume $D(R,\mathbf{S})$ shows the set of jump discontinuities of the resource profiles in schedule $\mathbf{S}$, that is, the set of points in time $t\in\{0,1,\ldots,\delta-1\}$ such that $r_k(\mathbf{S},t)\neq r_k(\mathbf{S},t-1)$. Assume es_i and ls_i indicate the earliest and the latest start times of activity i, respectively, calculated using the critical path method (CPM). If we consider $ES_i(\mathbf{S})$ as the set of eligible start times of activity i, we have $ES_i(\mathbf{S})=\{\{t|t\in([es_i,ls_i]\cap D(R,\mathbf{S}))\}\cup\{t|t\in[es_i,ls_i]\text{ and }t+d_i\in D(R,\mathbf{S})\}\cup\{es_i,ls_i\}\}$. Neumann and Zimmermann (1999) proved that there is an optimal schedule $\mathbf{S}$ in which $s_i\in ES_i(\mathbf{S})$ for all $i\in N$.

For instance, consider the example project depicted in Figure 1 in which $n=5$, $m=1$ and $\delta=14$. The number shown above each node (activity) indicates the activity duration while the below number specifies resource requirement of the activity.

Figure 2 shows a non-optimal schedule because $s_5\notin ES_5$ where ES_5 is calculated as follows:

$$\begin{aligned}ES_5&=\{\{t|t\in([3,9]\cap\{0,2,3,10,14\})\}\cup\{t|t\in[3,9]\text{ and }\\&\quad t+5\in\{0,2,3,10,14\}\}\cup\{3,9\}\}\\&=\{\{3\}\cup\{5,9\}\cup\{3,9\}\}=\{3,5,9\}\end{aligned}$$

3.2. General overview

Path-relinking (PR) is an intensification strategy to explore trajectories connecting elite solutions obtained by heuristic

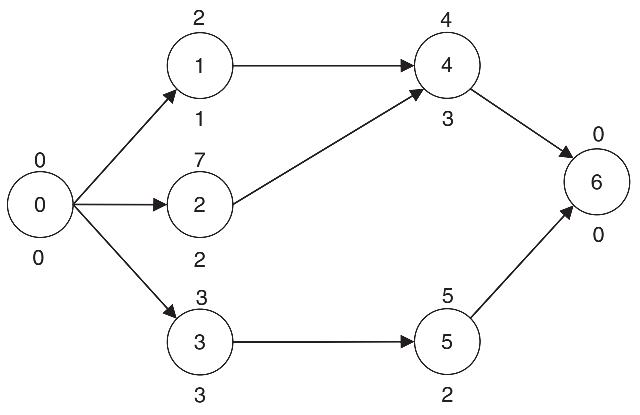

Figure 1 Example project.

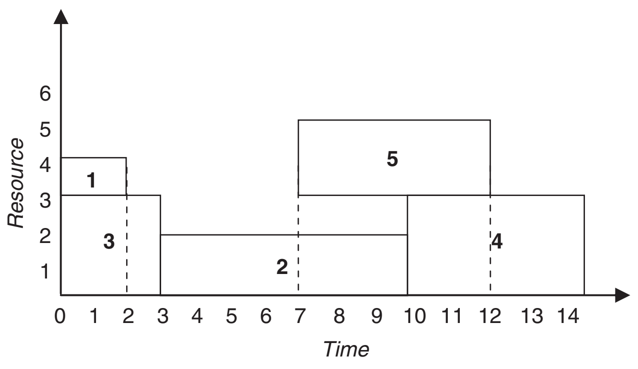

Figure 2 A non-optimal schedule.

methods (Glover, 1996). Path-relinking can be considered as an extension of the combination method of scatter search (Laguna and Martí, 2003). Instead of directly producing a new solution when combining two or more original solutions, path-relinking generates paths between and beyond the selected solutions in the neighbourhood space. It should be noted that the combination method in scatter search is a problem-dependent element, which is customized depending on the problem and the solution representation. In particular, in global optimization, where solutions are represented as real vectors, most scatter search applications perform linear combinations between pairs of solutions. Alternatively, in problems where solutions are represented as permutations, such as ordering problems, voting methods have been widely applied. In problems where solutions are represented as binary vectors, such as knapsack problems, probabilistic scores have provided very good results (Laguna and Martí, 2003). This way, one can also view path-relinking as a unified combination method for all types of problems and in this way it also generalizes the combination methods. This approach generates new solutions by searching paths that connect high-quality solutions by starting from one of these solutions, called an *initiating solution*, and generating a path in the neighbourhood space that leads towards the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions.

Figure 3 shows the main structure of our algorithm as pseudocode. In the first step, an initial population P containing $|P|$ solutions is generated using *diversification generation method*, described in Section 3.3. In the second step, the reference set *RefSet*, including b elite solutions of P, is constructed using *reference set building method*, described in Section 3.4. The solutions of *RefSet* are called reference solutions. Next, the *NewSubsets*, each of them containing two reference solutions, are generated. Subsequently, the two solutions of each subset, shown by $(\mathbf{S}', \mathbf{S}'')$, are selected and the *relinking method*, described in Section 3.5, is applied to them. We consider $\mathbf{S}'$ as the initial solution and $\mathbf{S}''$ as the guiding solution.

The relinking method generates solutions $\mathbf{S}'(1), \mathbf{S}'(2), \ldots, \mathbf{S}'(l)$ in the path constructed from $\mathbf{S}'$ towards $\mathbf{S}''$ in which $\mathbf{S}'(i)$ differs from $\mathbf{S}'(i+1)$; $i=1, \ldots, l-1$ in start time of only one activity. From the newly generated solutions in step 6, only one solution should be a candidate for adding to P. Since a *repairing method*, described in Section 3.6, and an *improvement method*, described in Section 3.7, are applied to selected solutions (step 9) and relative quality of solutions may be changed after repairing and improvement, two solutions are chosen randomly from generated solutions in step 6, shown by $\mathbf{S}'(u)$ and $\mathbf{S}'(v)$ where $u \neq v$. In step 8, the solutions that are not subject to the precedence constraints or the optimal property are repaired. Also, in step 9, a local search procedure is applied to the solutions as an improvement method. Next, two selected solutions are evaluated using evaluation function and the better one, shown by $\widehat{\mathbf{S}}$, is chosen. The evaluation function calculates $f(\mathbf{S})$ for a given solution $\mathbf{S}$. In step 11, $\widehat{\mathbf{S}}$ is added to P. In step 12, we exchange initial and guiding solutions and repeat steps 6–11 again. It should be noted that due to the structure of the relinking method, number of generated solutions in two opposite paths between $\mathbf{S}'$ and $\mathbf{S}''$ are identical. In order to keep the best solution so far, we add it to the P. Finally, we check termination criterion, the specified time limit, and go to step 2 if it is not met. In the next iterations, we do not need to construct an initial population because it has been generated in previous iteration. In other words, by combining each pair of *RefSet* (referred to as a *NewSubset*), two new solutions are added to the population.

3.3. Diversification generation method

Each element in the initial population is a random solution and is created by the diversification generation method, illustrated in Figure 4. In this approach, activities are added one by one to a partial schedule $\mathbf{S}$. $EA(\mathbf{S})$ indicates set of eligible activities, initialized by all activities without

```
1. Construct an initial population $P$ with size of $|P|$ using diversification generation method.
2. Build the RefSet using the reference set building method.
3. Generate NewSubsets with the subset generation method. Make $P=\emptyset$.
4. while (NewSubsets$\neq\emptyset$) do
5.    Select a next pair $(\mathbf{S}', \mathbf{S}'')$ in NewSubsets and delete it from NewSubsets.
6.    Apply the relinking method to produce the sequence $\mathbf{S}' = \mathbf{S}'(1), \mathbf{S}'(2), \ldots, \mathbf{S}'(l) = \mathbf{S}''$
7.    Select two solutions $\mathbf{S}'(u)$ and $\mathbf{S}'(v)$ $(u \neq v)$ from generated solution in step 6 randomly.
8.    Apply the repairing method to $\mathbf{S}'(u)$ and $\mathbf{S}'(v)$.
9.    Apply the improvement method to $\mathbf{S}'(u)$ and $\mathbf{S}'(v)$.
10.   Evaluate improved solutions and let the better one as $\hat{\mathbf{S}}$.
11.   let $P = P \cup \hat{\mathbf{S}}$.
12.   Exchange $\mathbf{S}'$ and $\mathbf{S}''$ and repeat steps 6 to 11.
13. end while
14. Add the best solution so far to the $P$.
15. If termination criterion is met, stop; otherwise, go to step2.
```

Figure 3 pseudocode of path-relinking method.

```
1. Let $EA(\mathbf{S}) = \{i | i \in N, Pred(i) = \emptyset\}$ and $SA(\mathbf{S}) = \emptyset$.
2. while $EA(\mathbf{S}) \neq \emptyset$ do
3.    Select activity $j \in EA(\mathbf{S})$ randomly.
4.    Calculate $ES_j(\mathbf{S})$, select $t \in ES_j(\mathbf{S})$ randomly and let $s_j = t$
6.    Update $EA(\mathbf{S})$ and $SA(\mathbf{S})$
7. end while
```

Figure 4 pseudocode of diversification generation method.

predecessor, and $SA(\mathbf{S})$ represents set of scheduled activities, initialized as an empty set.

After initialization step, we follow a while-loop to construct a random solution. In each iteration of this loop, an individual activity is added to the partial schedule $\mathbf{S}$. At step 3, one of the eligible activities, shown by j, is selected randomly. Next, $ES_j^S(\mathbf{S})$ is calculated, one of the eligible events for start time of activity j is chosen randomly and activity j is started in that time. Subsequently, sets $EA(\mathbf{S})$ and $SA(\mathbf{S})$ are updated. In order to update set $EA(\mathbf{S})$, activity j should be removed from $EA(\mathbf{S})$ and all activities $i \in Suc(j)$ for which $Pred(i) \subseteq SA(\mathbf{S})$ should be added to $EA(\mathbf{S})$, where $Suc(j)$ and $Pred(j)$ indicate the set of successors and set of predecessors of activity j, respectively. Also, set $SA(\mathbf{S})$ is updated simply as $SA(\mathbf{S}) = SA(\mathbf{S}) \cup j$.

Finally, it should be noted that each solution in our algorithm is represented by the vector of its start times. Although this type of solution representation, direct representation, is not recommended in metaheuristic algorithms due to vast range for possible values of start times, our solutions representation is efficient because in our algorithm, start time of each activity should be subject to precedence relations as well the optimal property described in Section 3.1.

3.4. Reference set building and subset generation methods

The reference set, *RefSet*, is a collection of both high-quality solutions and diverse solutions that are used to generate new solutions by way of applying the relinking method. We construct *RefSet* based on the method applied for construction of *RefSet*$_1$ of scatter search in Ranjbar *et al* (2009). We select the solution with smallest objective function, shown as $\mathbf{S}_1$, as the first member of *RefSet* and delete it from P. The next best solution $\mathbf{S}$ in P is chosen and added to *RefSet* only if $D_{\min}(\mathbf{S}) \geqslant th\_dist$, where $D_{\min}(\mathbf{S})$ is the minimum of the distances of solution $\mathbf{S}$ to the solutions currently in *RefSet* and *th_dist* is a threshold distance. The difference between two solutions is number of different start times for identical real activities in two solutions divided by number of real activities. Thus, the difference of every two solutions changes in range [0, 1]. This process is repeated until b members are chosen for *RefSet*. Whenever no qualified solution can be found in the population, the *RefSet* is completed with random solutions generated using diversification generation method (Ranjbar *et al*, 2009). For these members of *RefSet*, the condition of minimum threshold distance is ignored.

After the *RefSet* construction, *NewSubsets* are generated, consisting of all pairs of reference solutions resulting in $(b^2-b)/2$ *Newsubsets*. The pairs in *NewSubsets* are selected one at a time in lexicographical order and the relinking method is applied to generate one trial solution. Thus, size of P will be always $(b^2-b)/2+1$ where one more solution shows the best solution so far.

3.5. Relinking method

The relinking method constructs a path from initial solution towards guiding solution. This path is built of a sequence of elements in which each element is a vector of start times and differs with its previous and next elements in start time of only one activity.

Assume the relinking method should be applied to two given solutions $\mathbf{S}'=(s'_0,\ldots,s'_{n+1})$ and $\mathbf{S}''=(s''_0,\ldots,s''_{n+1})$, where the path direction is from $\mathbf{S}'$ towards $\mathbf{S}''$. The relinking method is an iterative procedure with at most n iterations. At the ith iteration of this procedure, s''_i is compared with s'_i and if $s''_i \neq s'_i$, we change s'_i as $s'_i = s''_i$. After each change, a new solution is obtained. It should be noted that generated solutions may not observe precedence constraints or the optimal property. Thus, selected solutions of each path should follow the repairing method.

For instance, consider the example project depicted in Figure 1 and assume schedules $\mathbf{S}'$ and $\mathbf{S}''$ are given as $\mathbf{S}'=(0,3,0,0,7,7,11)$ and $\mathbf{S}''=(0,0,3,0,10,3,14)$. If we suppose moving direction is from $\mathbf{S}'$ towards $\mathbf{S}''$, the generated solutions in this path are shown in Figure 5. Since $s''_1 \neq s'_1$, $\mathbf{S}'(1)$ is generated from $\mathbf{S}'$ by changing s'_1 as $s'_1 = s''_1 = 0$. Similarly, $\mathbf{S}'(2)$ is generated from $\mathbf{S}'(1)$ by changing $s'_2 = s''_2 = 3$. It should be noted that in $\mathbf{S}'(3)$, s'_4 is changed but the project termination time (s'_{n+1}) is changed automatically to 14.

Initial Solution	$\mathbf{S}'$ = (0,3,0,0,7,7,12)
Generated solutions	$\mathbf{S}'(1)$ = (0,0,0,0,7,7,12)
	$\mathbf{S}'(2)$ = (0,0,3,0,7,7,12)
	$\mathbf{S}'(3)$ = (0,0,3,0,10,7,14)
Guiding solution	$\mathbf{S}''$ = (0,0,3,0,10,3,14)

Figure 5 Illustration of the relinking method.

3.6. Repairing method

The repairing method includes two steps. The first step is applied to infeasible solutions for the purpose of making them feasible while the second step is applied to the solutions that do not have the optimal property. In the first step in which the precedence constraints are investigated, for each activity $i \in N$, if $s_i < \max_{j \in Pred(i)} \{f_j\}$, then we set $s_i = \max_{j \in Pred(i)} \{f_j\}$. In the second step of the repairing method, for each activity $i \in N$ that $s_i \notin ES_i$, we shift the activity to direction that has more positive or less negative impact on the objective function by the minimum amount required such that we have $s_i \in ES_i$. It has been assumed that for each $(i,j) \in A$ the relation $i<j$ is held. Thus, in order to terminate the repairing method in a finite number of iterations, we apply both the first and the second steps of the repairing method in ascending order of the activity numbers.

For instance, consider the example project and assume $\mathbf{S}'(2)=(0,0,3,0,7,7,12)$, generated in the previous section, is selected for repairing. Since activity 2 is started at time instant 3 and finished at time instant 10, activity 4 cannot be started at a time instant less than 10. Thus, at the first step of the repairing method, we change s'_4 from 7 to 10. The output of the first step of the repairing method is the schedule depicted in Figure 2 in which $s'_5 \notin ES_5$. So, in the second step of the repairing method, s'_5 is changed from 7 to 5.

3.7. Improvement method

The improvement method is a local search procedure that gets a schedule $\mathbf{S}$ as input and searches in the neighbourhood of $\mathbf{S}$ in the hope of finding a better solution. The strategy of this local search is the *best improvement* strategy, that is, if more than one better solution is found in the neighbourhood of a given solution, the best one is chosen. If $\mathbf{S}^*$ shows the best found solution in the neighbourhood of $\mathbf{S}$ and $f(\mathbf{S}^*)<f(\mathbf{S})$, the improvement method is restarted by letting $\mathbf{S}=\mathbf{S}^*$. This procedure is repeated until no improvement is obtained.

For a given schedule $\mathbf{S}$, we first calculate the jump for each $t \in D(R,\mathbf{S})$. Then, we select a set of candidate activities for which the jump in their start or finish time is more than average of jumps. Next, for each candidate activity we generate its neighbours as follows. First of all, for each candidate activity i, we calculate $ES_i(\mathbf{S})$. Then, for each $t \in ES_i(\mathbf{S})$ there is an iteration in which we set $s_i = t$ while start times of other activities are not changed. After that, s_i is set to its initial value, specified by schedule $\mathbf{S}$, and next candidate activity is chosen for changing its start time. This procedure is repeated for all candidate activities and each change in the start time of each individual activity generates one of the neighbours of schedule **S**. In order to improve generated neighbours, we apply the repairing method to each neighbour. Since start time of each activity $i \in N$ is changed in the domain specified by $ES_i(\mathbf{S})$, generated neighbours are feasible in terms of precedence constraints but they may not obey the optimal property. Thus, we apply only the second step of the repairing method to generated neighbours.

For instance, consider the example project in which the improvement method is applied to the schedule $\mathbf{S}'(2) = (0, 0, 3, 0, 7, 5, 14)$, obtained from repairing method in the previous section. The only improvement is obtained if s'_1 is changed from 0 to 3. The resulting schedule $\widehat{\mathbf{S}} = (0, 3, 3, 0, 7, 5, 14)$, which is also optimal for the example project, is depicted in Figure 6.

4. Performance analysis

We have performed a series of computational experiments using randomly generated test sets in order to examine the performance of the *PR* algorithm.

4.1. Experimental setup

A test set including 600 test instances were generated using the random network generator RanGen (Demeulemeester *et al*, 2003). Five different values $n = 100, 200, 300, 400$ and 500 have been chosen as the number of activities. Also, the values of *order strength* (*OS*) have been considered as $OS = 0.25, 0.5$ and 0.75. The *OS* parameter indicates the number of precedence-related activity pairs divided by the theoretically maximum number of such pairs in the network. In addition, five resources are considered and the resource demands and coefficients c_k are generated randomly from the discrete uniform distribution $\{1, \ldots, 10\}$. For each combination of n and *OS*, 10 random test instances have been generated. Besides n and *OS*, the prescribed project deadline δ is an important parameter when dealing with the RLP. For each test instance, four different values $\delta = es_{n+1}, 1.1es_{n+1}, 1.25es_{n+1}$ and $1.5es_{n+1}$ have been considered where es_{n+1} represents the earliest start time of dummy end activity $n+1$.

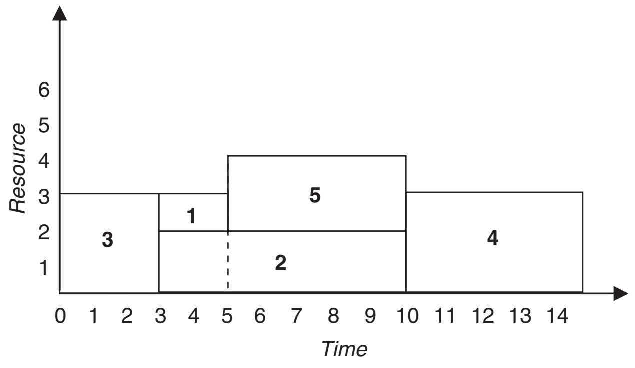

Figure 6 Illustration of the improvement method.

We used a full factorial design of experiments to determine the best values of parameters b and *th_dist*. For this purpose, we consider four values $n/2$, n, $3n/2$ and $2n$ for parameter b and four values $0.2, 0.4, 0.6$ and 0.8 for parameter *th_dist*. For each combination, we consider only a small set of hard test instances, including 50 test instances, to measure the performance. The best performance of the *PR* algorithm is obtained when $b = n$ and $th\_dist = 0.4$.

4.2. Computational results

In order to evaluate the efficiency of our developed *PR* metaheuristic, one way is to compare it with similar approaches from literature. In particular, the tabu search (*TS*) algorithm of Neumann and Zimmermann (2000) and iterated greedy (*IG*) algorithm of Ballestin *et al* (2007) are considered. We reimplemented *TS* and *IG* algorithms according to the settings described in the corresponding papers. All codings were done in the Visual C++ 6.0 environment. In order to verify our implementations, we first found computers similar to those used for running *TS* and *IG* algorithms (PC Pentium with 200 processor running NT 4.0 for *TS* algorithm and PC Pentium with 1.4GHz clock pulse and 512MB RAM for *IG* algorithm). Statistical tests, not reported in this paper, indicated that there is no difference between results obtained by our implementations and the results reported in the two foregoing papers. Then, for a fair comparison, we ran all algorithms on PC Pentium IV 3 GHz processor with 1024 MB of internal memory and we selected three different time limits $TL = 10, 30$ and 60 s as the termination criterion.

The minimum objective function for each test instance among all runs of the three algorithms is considered as the *best found solution*. Table 1 shows the average percent deviation of solutions found by the *PR* from the best found solutions. Similar to Table 1, Tables 2 and 3 show the same information but found by the *TS* and *IG* algorithms, respectively.

As expected, by increasing the run time limit, the quality of the obtained solutions is improved. Also, the average percent deviation from best found solutions is higher for the projects with larger number of activities. This trend is consistent for all three algorithms and all the run time limits except the case of the *PR* algorithm and $TL = 60$ s. The reason of this exception is that for larger projects, most of the best solutions are found by the *PR* algorithm.

The results of Tables 1–3 reveal that *PR* outperforms two other metaheuristics. In the *PR* algorithm, the average

Table 1 Average percent deviation of the solutions found by the *PR* from the best found solutions

	Time Limit (in seconds)		
	10	30	60
$n=100$	36.1	25.0	3.1
$n=200$	37.1	29.6	3.9
$n=300$	47.8	30.7	2.0
$n=400$	50.9	35.9	1.4
$n=500$	53.5	38.8	0.4

Table 2 Average percent deviation of the solutions found by the *TS* from the best found solutions

	Time Limit (in seconds)		
	10	30	60
$n=100$	48.7	26.8	8.3
$n=200$	58.4	38	12.9
$n=300$	58.8	41.5	33.0
$n=400$	67.5	47.3	23.2
$n=500$	73.4	49.0	21.7

Table 3 Average percent deviation of the solutions found by the *IG* from the best found solutions

	Time Limit (in seconds)		
	10	30	60
$n=100$	40.2	19.9	5.9
$n=200$	44.1	29.8	8.4
$n=300$	53.7	36.7	13.4
$n=400$	61.2	41.2	18.9
$n=500$	65.4	44.0	20.7

percent deviation for all test instances with run time limits 10, 30 and 60 s is around 45, 32 and 2%, respectively. These total average percent deviation values are 61, 40 and 19% for the *TS* algorithm and 52, 34 and 13% for the *IG* algorithm. Comparison of the results of the algorithms *TS* and *IG* convinced us that *IG* is more efficient than *TS*.

Table 4 shows the percent of average relative deviation between the objective function values provided by the *TS* and the *PR* for each combination of n and δ while TL is set to 30 s. Similar comparison is shown in Table 5 between the *IG* and the *PR* algorithms.

The results of Tables 4 and 5 confirm the obtained results from Tables 1–3. There is a constant trend in both Tables 4 and 5 in which the relative efficiency of the *PR* algorithm than two other algorithms is more revealed when values of δ and n are increased.

Table 4 Percent of mean relative deviation between objective function values of the *TS* and *PR*

	$\delta=es_{n+1}$	$\delta=1.1es_{n+1}$	$\delta=1.25es_{n+1}$	$\delta=1.5es_{n+1}$
$n=100$	2.8	8.0	13.3	20.4
$n=200$	3.1	8.4	13.5	20.3
$n=300$	3.4	9.1	13.9	21.0
$n=400$	3.6	9.0	13.1	11.8
$n=500$	3.7	8.5	12.8	20.7

Table 5 Percent of mean relative deviation between objective function values of the *IG* and *PR*

	$\delta=es_{n+1}$	$\delta=1.1es_{n+1}$	$\delta=1.25es_{n+1}$	$\delta=1.5es_{n+1}$
$n=100$	1.2	3.7	6.0	9.1
$n=200$	1.3	4.2	6.9	9.7
$n=300$	1.7	4.4	7.3	10.1
$n=400$	1.7	4.8	7.5	10.8
$n=500$	1.7	4.5	7.3	10.3

Table 6 Percent of mean relative deviation between objective function values of the *PR* with and without the improvement method

	$\delta=es_{n+1}$	$\delta=1.1es_{n+1}$	$\delta=1.25es_{n+1}$	$\delta=1.5es_{n+1}$
$n=100$	20.3	23.8	28.7	33.1
$n=200$	21.4	27.8	32.1	35.8
$n=300$	25.8	31.1	36.1	38.4
$n=400$	28.5	35.9	39.4	44.4
$n=500$	34.0	39.9	44.7	47.3

4.3. Impact of the improvement method

In order to evaluate the impact of the improvement method (local search), the *PR* algorithm is run without the improvement method over all test instances. Table 6 displays the average relative deviation of the objective function values obtained by the *PR* without and with the improvement method while $TL=30$ s. The average percent improvement is noticeable, that is, 26, 31.7, 36.2 and 39.8% for $\delta=es_{n+1}$, $\delta=1.1es_{n+1}$, $\delta=1.25es_{n+1}$ and $\delta=1.5es_{n+1}$, respectively. These results indicate that the developed improvement method is efficient.

5. Summary and outlook on further research

In this article, the RLP is studied. A dedicated path-relinking metaheuristic is developed for the problem and is compared with two best metaheuristics in the literature. The comparative computational results reveal that the developed path-relinking metaheuristic procedure is efficient and outperforms two other algorithms.

Future research should be oriented towards applying the path-relinking metaheuristic for other project scheduling problems. Also, developing other metaheuristic for the RLP can be interesting research topics.

Acknowledgements—This work was supported in part by Ferdowsi University of Mashhad's Research Council, under grant #18203 (dated 28-6-2011).

References

Ahuja HN (1976). *Construction Performance Control by Networks*. Wiley: New York.

Ballestin F, Schwindt C and Zimmermann J (2007). Resource leveling in make-to-order production: Modeling and heuristic solution method. *International Journal of Operations Research* **4**(1): 50–62.

Bandelloni M, Tucci M and Rinaldi R (1994). Optimal resource leveling using non-serial dynamic programming. *European Journal of Operational Research* **78**(2): 162–177.

Brinkmann K and Neumann K (1996). Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags: The resource-levelling and minimum project-duration problems. *Journal of Decision Systems* **5**(1–2): 129–155.

Burgess AR and Killebrew JB (1962). Variation of activity level on a cyclic arrow diagram. *Journal of Industrial Engineering* **13**(2): 76–83.

Demeulemeester E (1995). Minimizing resource availability costs in time-limited project networks. *Management Science* **41**(10): 1590–1598.

Demeulemeester E and Herroelen W (2002). *Project Scheduling—A Research Handbook*. Kluwer Academic Publishers: London.

Demeulemeester E, Vanhoucke M and Herroelen W (2003). RanGen: A random network generator for activity-on-the-node networks. *Journal of Scheduling* **6**(1): 13–34.

Drotos M and Kis T (2011). Resource leveling in a machine environment. *European Journal of Operational Research* **212**(1): 12–21.

Easa SM (1998). Resource leveling in construction by optimization. *Journal of Construction Engineering and Management* **115**(2): 302–316.

Geng JQ, Weng LP and Liu SH (2011). An improved ant colony optimization algorithm for nonlinear resource-leveling problems. *Computers and Mathematics with Applications* **61**(8): 2300–2305.

Glover F (1996). Tabu search and adaptive memory programming—Advances, applications and challenges. In: Barr RS, Helgason RV and Kennington JL (eds). *Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers: London, pp 1–75.

Harris RB (1978). *Precedence and Arrow Networks for Construction*. Wiley: New York.

Harris RB (1990). Packing method for resource leveling (pack). *Journal of Construction Engineering and Management* **116**(2): 331–350.

Hartmann S and Briskorn D (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research* **207**(1): 1–14.

Laguna M and Martí R (2003). *Scatter Search: Methodology and Implementations in C*. Operations Research/Computer Science Interfaces Series Kluwer Academic Publishers: Boston, MA.

Leu SS, Yang CH and Huang JC (2000). Resource leveling in construction by genetic algorithm-based optimization and its decision support system application. *Automated Construction* **10**(1): 27–41.

Meridith JR and Mantel SJ (1995). *Project Management: A Managerial Approach*. Wiley: New York.

Moder JJ, Phillips CR and Davis EW (1983). *Project Management with CPM PERT and Project Diagramming*. Van Nostrand Reinhold: New York.

Neumann K and Zimmermann J (1999). Resource leveling for projects with schedule-dependent time windows. *European Journal of Operational Research* **117**(3): 591–605.

Neumann K and Zimmermann J (2000). Procedures for resource levelling and net present value problems in project scheduling with general temporal and resource constraints. *European Journal of Operational Research* **127**(2): 425–443.

Neumann K, Schwindt C and Zimmermann J (2003). *Project Scheduling with Time Windows and Scarce Resources*. Springer: Berlin.

Ranjbar M, De Reyck B and Kianfar F (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. *European Journal of Operational Research* **193**(1): 35–48.

Roca J, Pugnaghi E and Libert G (2008). Solving an extended resource leveling problem with multiobjective evolutionary algorithms. *International Journal of Information and Mathematical Sciences* **4**(4): 289–300.

Savin D, Alkass S and Fazio P (1996). Construction resource leveling using neural networks. *Canadian Journal of Civil Engineering* **23**(4): 917–925.

Savin D, Alkass S and Fazio P (1997). A procedure for calculating the weight-matrix of a neural network for resource leveling. *Advanced Engineering Software* **28**(5): 277–283.

Takamoto M, Yamada N, Kobayashi Y and Nonaka H (1995). Zero-one quadratic programming algorithm for resource leveling of manufacturing process schedules. *Systems and Computers in Japan* **26**(10): 68–76.

Tavares LV (1987). Optimal resource profiles for program scheduling. *European Journal of Operational Research* **29**(1): 83–90.

Younis MA and Saad B (1996). Optimal resource leveling of multi-resource projects. *Computers and Industrial Engineering* **31**(1–2): 1–4.

Received October 2011;
accepted August 2012 after two revisions