International Journal of Industrial Engineering & Production Research

september 2012, Volume 23, Number 3

pp. 231-243

http://IJIEPR.iust.ac.ir/

ISSN: 2008-4889

A Hybrid GRASP Algorithm for Minimizing Total Weighted Resource Tardiness Penalty Costs in Scheduling of Project Networks

M. Ranjbar*

Mohammad Ranjbar, *Assistant Professor, Ferdowsi University of Mashhad, Department of Industrial Engineering*

KEYWORDS

Project scheduling;
Weighted Resource tardiness;
GRASP;
path-relinking

ABSTRACT

In this paper, we consider scheduling of project networks under minimization of total weighted resource tardiness penalty costs. In this problem, we assume constrained resources are renewable and limited to very costly machines and tools which are also used in other projects and are not accessible in all periods of time of a project. In other words, there is a dictated ready date as well as a due date for each resource such that no resource can be available before its ready date but the resources are allowed to be used after their due dates by paying penalty costs depending on the resource type. We also assume, there is only one unit available of each resource type and no activity needs more than it for execution. The goal is to find a schedule with minimal total weighted resource tardiness penalty costs. For this purpose, we present a hybrid metaheuristic procedure based on the greedy randomized adaptive search algorithm and path-relinking algorithm. We develop reactive and non-reactive versions of the algorithm. Also, we use different bias probability functions to make our solution procedure more efficient. The computational experiments show the reactive version of the algorithm outperforms the non-reactive version. Moreover, the bias probability functions defined based on the duration and precedence relation characteristics give better results than other bias probability functions.

1. Introduction

The goal of the resource-constrained project scheduling problem (RCPSP) is to minimize the duration of a project subject to finish-to-start type precedence constraints and renewable resources constraints. It is shown in Blazewicz et al. [1] that the RCPSP, as a generalization of the job-shop scheduling problem, is NP-hard in the strong sense. A large number of exact and heuristic procedures have been proposed to construct workable baseline schedules for this problem; see Demeulemeester and Herroelen [2], Neumann et al. [3] for recent overviews and Herroelen [4] for a discussion on the link between theory and practice.

In some projects, some expensive resources like especial types of crane, tunnel boring machines, very expert humans and etc. are often hired out of the project. Companies that lease these costly resources

* **Corresponding author: Mohammad Ranjbar**

Email: *m_ranjbar@um.ac.ir*

Paper first received Sep. 28, 2011, and in revised form June 16, 2012.

have a plan for leasing them and consequently this schedule dictates ready dates and due dates to the customers.

We assume these types of resources are constrained renewable and are not available in all periods of time of a project horizon. We also assume only these resources are constrained while other resources are unlimited. In most of the projects, usually one unit of each expensive resource type is hired and no activity needs more than it for execution. For each resource type, we consider a ready date, a due date and a penalty cost.

No resource can be accessible before its ready date but these resources are permitted to be released after their due dates by paying penalty costs. The goal is to find a schedule with minimal total weighted resource tardiness penalty costs. Thus, we face to a RCPSP under minimization of total weighted resource tardiness penalty cost, shown by the RCPSP-TWRTPC. The RCPSP-TWRTPC was introduced by Ranjbar et al. [5] and they presented a branch-and-bound algorithm for it.

Also, Khalilzadeh et al. (2012) introduced a generalized version of the RCPSP-TWRTPC in which the multi mode projects are considered. They developed a particle swarm metaheuristic for the proposed problem.

Other related scheduling problems in the literature are in fields of project scheduling and machine scheduling with objective functions linked to the tardiness. In all of these problems, the issue of tardiness is proposed for activities or jobs and not for resources or machines. Vanhoucke et al. [7] have developed a branch-and-bound (B&B) algorithm accompanied with an exact recursive search procedure for the RCPSP under earliness/tardiness objective. Also, Nadjafi and Shadrokh [8] developed a B&B algorithm for the weighted earliness-tardiness project scheduling problem with generalized precedence relations.

The contributions of this article are two: (1) we develop reactive and nonreactive versions of a hybrid metaheuristic for the RCPSP-TWRTPC; (2) we develop seven biased probability functions to make algorithm more efficient and show, using computational results, that biased probability functions defined on the basis of duration and precedence relation characteristics outperforms others.

The remainder of this article is organized as follows. Problem modeling and formulation are provided in Section 2. Section 3 presents our solution representation while Section 4 is devoted to our developed metaheuristic algorithm. The computational experiments are presented in Section 5. Finally, summary and conclusions are given in Section 6.

2. Problem Modeling and Formulation

The RCPSP-TWRTPC can be represented by a disjunctive graph $G=(N,C,D)$. Graph G has an activity-on-node (AON) representation in which $N=\{0,1,...,n+1\}$ indicates the set of activities (nodes) where dummy activities 0 and n+1 represent start and end of the project. The set of conjunctive arcs $C=\{(i,j);i\rightarrow j,i,j\in N\}$ consists of arcs representing technical finish-to-start type precedence relations among activities where $i\rightarrow j$ implies activity j can be started after finishing of activity i. Let $R=\{1,2,...,m\}$ be the set of constrained renewable resources and N_r the set of activities which need (one unit of) resource $r\in R$ for execution. For each pair of activities $i,j\in N_r; r=1,...,m$, there is a disjunctive arc $i\leftrightarrow j$ between nodes i and j requiring the resource r. Thus, we present a set of disjunctive arcs as $D=\{\langle i,j\rangle; i\leftrightarrow j, \exists r\in R: i,j\in N_r\}$ such that we should determine $i\rightarrow j$ or $j\rightarrow i$ because availability of each resource is at most one unit in each period of time and two activities i and j where $\langle i,j\rangle\in D$ can not be processed in parallel. For each activity i, the parameter d_i indicates its duration where $d_0=d_{n+1}=0$. In addition, for each resource r, ρ_r, δ_r and w_r show the ready date, due date and weight of this resource, respectively. In order to embed the resource ready dates in the graph representation, we add one node corresponding to each resource to the project network.

For the resource r, this node displays an activity with duration ρ_r which is direct successor of the start dummy activity and direct predecessor of every activity $i\in N_r$. Also, we consider these arcs as the elements of the set of conjunctive arcs C.

Table 1 shows the resource information of a RCPSP-TWRTPC instance with n=6 real activities and m=2 resources while the corresponding graph is depicted in Figure 1 (Ranjbar et al. [5]). In this figure, the number shown above each node indicates activity duration and the number(s) below indicate the resources required for activity execution. The nodes labeled α and β correspond to ready times of resources 1 and 2, respectively. Precedence relations of each of these nodes with dummy node 0 and the nodes which require resources 1 and 2 are depicted with bold arcs. Also, the disjunctive arcs are depicted with dashed lines while conjunctive arcs are shown as regular arcs. Any solution of a RCPSP-TWRTPC instance is shown by the vector $\mathbf{S}=(s_1,s_2,...,s_n)$ where s_i is an non-negative integer and shows the start time of activity i. Given a policy for scheduling, such as earliest start time schedule, this solution $\mathbf{S}$ is equivalent to a selection $\sigma(D)$, denoting a selection of disjunctive arcs from D, as long as the selection $\sigma(D)$ has one and only one arc from every pair $i\leftrightarrow j$, and the resulting

graph $G=(N,C,\sigma(D))$ is not cyclic. Conversely, any selection $\sigma(D)$ satisfying the above properties corresponds to a feasible schedule.

Let $L(i, j)$ denote the length of the longest path from node *i* to node *j* in graph $G=(N,C,\sigma(D))$ (if there is no path between *i* and *j*, then $L(i, j)$ is not defined). The (earliest) finish time of activity *i*, $f_i = s_i + d_i$, equals to $L(0,i)$ and can be computed using the algorithm of Bellman [9] with complexity $O|N|$.

The release time of resource *r* shown by c_r equals to $c_r = \max_{i \in N_r}\{f_i\}$ and the tardiness of this resource is calculated as $T_r = \max\{c_r - \delta_r, 0\}$. Then, the total weighted resource tardiness penalty cost equals to $\sum_{r=1}^{m} w_r T_r$.

Tab.1. Resource information of the example project

Resource (*r*)	ρ_r	δ_r	w_r	N_r
1	1	18	3	{1,2,4,5}
2	4	22	4	{3,4,5,6}

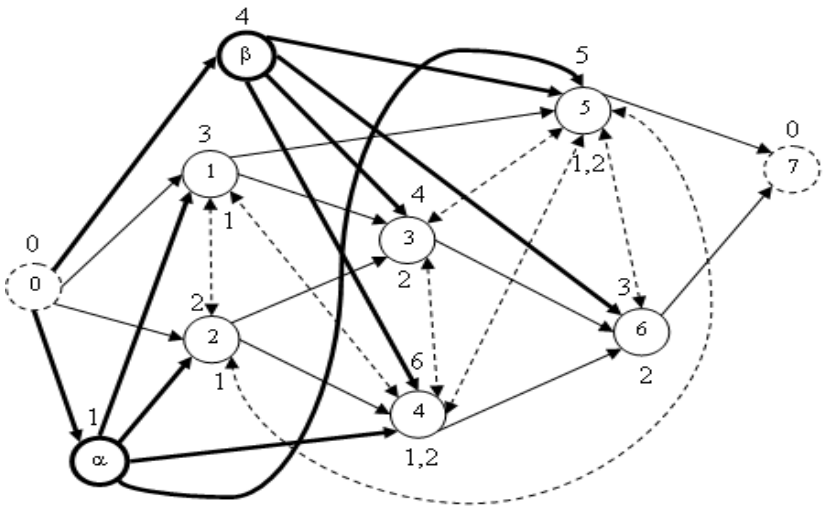

Fig.1. Graph of the example project

The RCPSP-TWRTPC can be formulated as the following linear integer programming model using variables s_i, c_r, T_r and X_{ij} where for all $\langle i, j\rangle \in D$, $X_{ij}=1$ if $i \leftrightarrow j = i \rightarrow j$ and $X_{ij}=0$ if $i \leftrightarrow j = j \rightarrow i$.

$$\min Z = \sum_{r=1}^{m} w_r T_r \quad (1)$$

Subject to:

$$c_r \geq s_i + d_i;\ r = 1,...,m;\ i \in N_r \quad (2)$$

$$T_r \geq c_r - \delta_r;\ r = 1,...,m \quad (3)$$

$$T_r \geq 0;\ r = 1,...,m \quad (4)$$

$$s_i \geq \rho_r;\ r = 1,...,m;\ i \in N_r \quad (5)$$

$$s_j - s_i \geq d_i;\ \forall (i, j) \in C \quad (6)$$

$$s_j - s_i \geq d_i - M(1 - X_{ij});\ \forall \langle i, j\rangle \in D \quad (7)$$

$$s_i - s_j \geq d_j - M X_{ij};\ \forall \langle i, j\rangle \in D \quad (8)$$

$$s_i, c_r, T_r \in \mathbf{Z}^+;\ i = 0,1,...,n+1;\ r = 1,2,...,m \text{ and } \forall \langle i, j\rangle \in D : X_{ij} \in \{0,1\} \quad (9)$$

The objective function (1) represents the minimization of the total weighted resource tardiness penalty costs. Constraint (2) shows that the release time of each resource is not less than the finish time of activities which require that resource. Constraints (3) and (4) ensure that T_r is equal to $\max\{c_r - \delta_r, 0\}$. Constraint

(5) makes the starting times of all activities greater than or equal to the ready dates of their corresponding resources. Constraint (6) represents the technical precedence relations or conjunctive constraints while constraints (7) and (8) relate to the resource or disjunctive constraints. Finally, constraint (9) in which $\mathbf{Z}^+$ indicates the set of non-negative integers, ensures that variables s_i , c_r and T_r are non-negative integers and X_{ij} is a binary variable.

3. Solution Representation

Our constructive heuristic algorithm uses a schedule representation to encode a project schedule and a schedule generation scheme to translate the schedule representation to a schedule **S**. In our problem, the schedule generation scheme determines how a feasible schedule is constructed by assigning starting times to the activities, whereby disjunctive arcs are converted to conjunctive arcs by schedule representation. We represent each solution of the RCPSP-TWRTPC using a binary list called direction list (*DL*) and shown by $DL = (e_1,...,e_{|D|})$. Each e_k in *DL* represents a direction for disjunctive arc $\langle i, j\rangle \in D$ and is a binary variable. It is one if we consider disjunctive arc $\langle i, j\rangle$ as conjunctive arc (i, j) and zero, otherwise. It should be noticed that to construct *DL*, we first sort elements of $D = \{\langle i, j\rangle\}$ on the basis of non-decreasing order of *i* in $\langle i, j\rangle$ (using smallest *j* as a tie-breaker). Then, e_k of *DL* relates to the k^{th} sorted $\langle i, j\rangle$, $k = 1,...,|D|$. For the example project, we have $D = \{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle, \langle 5,6\rangle\}$ and the optimal solution, found by enumeration, of this project is obtained with $DL = (e_1 = 1, e_2 = 1, e_3 = 1, e_4 = 0, e_5 = 0, e_6 = 1, e_7 = 1)$ corresponding to the following arcs: (1,2), (1,4), (2,5), (4,3), (5,3), (4,5) and (5,6). Each solution of the RCPSP-TWRTPC can be easily translated to a schedule **S** using the well-known critical path method (CPM), shown by **S**=CPM(*DL*). The optimal solution corresponding to the above mentioned *DL* is **S**=(1,4,17,6,12,21) with 8 units of tardiness penalty cost.

4. GRASP and Path-Relinking

Below, we discuss GRASP and path-relinking as a general heuristic procedure and describe the overall structure of our search procedure for resolution of the RCPSP-TWRTPC-solutions.

4-1. General Overview

In the following we briefly describe general GRASP and path-relinking procedures.

4-1-1. GRASP

A greedy randomized adaptive search procedure (GRASP) is a multi-start and iterative process (Aiex et al. [10]; Feo and Resende [11]; Feo et al. [12]). Each GRASP-iteration consists of two phases: in a construction phase, a feasible solution is produced and, in a local-search phase, a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. The basic construction phase in GRASP is similar to the semi-greedy heuristic proposed independently by Hart and Shogan [13]. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list with respect to a greedy function. This function measures the benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP resides in the fact that we choose one of the best candidates in the list but not necessarily the top candidate; the list of best candidates is called the restricted candidate list. It is almost always beneficial to apply a local-search procedure to attempt to improve each constructed solution.

4-1-2. Path-Relinking

Path-relinking is an enhancement to the basic GRASP procedure, leading to significant improvements in solution quality. Path-relinking was originally proposed by Glover [14] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search (see Glover and Laguna [15] and Glover et al. [16]). Starting from one or more elite solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Path-relinking may be viewed as a strategy that seeks to incorporate attributes of high quality solutions, by favoring these attributes in the selected moves.

The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Marti [17]. It was followed by several extensions, improvements, and successful applications (see Ribeiro et al. [18], Resennde et al. [19] and Alvarez et al. [20]).

4-2. Adapting GRASP and Path-Relinking to Our Setting

4-2-1. Global Structure of the Algorithm

The pseudo-code of global structure of our GRASP and path-relinking implementation is illustrated in Algorithm 1. Our basic algorithm maintains a set of elite solutions (*ES*) to combine them in step 9 using

path-relinking algorithm. This set is initializes as an empty set in the first step. A while-loop is repeated until termination criterion (*TC*), a specified number of iterations, is met. At the beginning of this loop, a *DL* is built using building direction list (*BDL*) procedure (Section 4.2.2). Next, generated *DL* is evaluated using CPM and is improved using local search (*LS*) procedure (Section 4.2.3).

In steps 6 to 11, we decide to add *DL* to the *ES* or not. For this purpose, we define *Max_Elite* as the maximum size of *ES* (size of *ES* is shown by |*ES*|) and $\Delta(\text{CPM}(DL),\text{CPM}(DL^i))$ as the difference between *DL* and DL^i, which is the number of different start times for identical activities in CPM(*DL*) and $\text{CPM}(DL^i)$ divided by *n*. The first condition for each *DL* to be included in *ES* is that it should be different from all elements in *ES*. This condition is checked in step 6 and if it is not satisfied, we go to the end of while loop at step 13 and discard generated *DL*. If the first condition is assured, the next condition is that the size of *ES* to be smaller than the *Max_Elite*. If *DL* is not added to *ES* in step 7, we should follow steps 8 to 12. In step 8, we select a direction list from *ES* on the basis of a biased random sampling strategy. Random sampling is biased using probability vector $\mathbf{Q}=(q_1,...,q_{Max\_Elite})$ in which $q_i = b_i \Big/ \sum_{j\in ES} b_j$ and $b_i = \Delta(\text{CPM}(DL),\text{CPM}(DL^i))/Z_i$ where DL^i is the i^{th} element of *ES* and Z_i stands for the value of the objective function for the solution $CPM(DL^i)$ where *i*=1,..,*Max_Elite*. For each DL^i, having smaller objective functions and higher differences with *DL* gives rise to its selection chance. The selected direction list, shown by DL', is combined with *DL* using path-relinking (*PR*) procedure developed by Ranjbar et al. [21] (Section 4.2.4). In step 11, we compare the output of *PR* procedure with the worst element of *ES*, shown by DL''. If DL' is better than DL'', it is replaced by DL''. Whenever *TC* is met, the best found solution is returned.

Algorithm 1: Global algorithm structure

1: *ES*=∅
2: **while** *TC* not met **do**
3: Build *DL* using *BDL*
4: **S**=CPM(*DL*)
5: $DL = LS(DL)$
6: **if** $\exists DL^i \in ES : \Delta(\text{CPM}(DL),\text{CPM}(DL^i)) = 0$ go to step 13.
7: **else if** $|ES| < Max\_Elite$ **then** $ES = ES \cup DL$
8: **else**
9: select DL' randomly from *ES* using probability vector **Q**
10: $DL = PR(DL, DL')$
11: **if** DL is better than the worst element DL'' in *ES* **then** $ES = (ES \setminus DL'') \cup DL$
12: **end else**
13: **end while**
14: Return the best found solution

4-2-2. Building Direction List Procedure

This is an iterative algorithm and in each iteration, at least one of the elements of $DL = (e_1,..., e_{|D|})$ is set to zero or one. For each unset element of *DL*, in each iteration, two candidate elements $\{0,1\}$ are defined in a candidate list *CL*.

Thus, at the beginning *CL* has $2\times|D|$ elements. In an iteration, let *u* as the number of unset elements of *DL*. Consequently, there are a total of *u* zeros and total of *u* ones in the corresponding *CL*. One of these zeros or ones is selected, as explained in the Algorithm 2. If the selected item is zero (one), then its corresponding element in *D* is set to zero (one). To assure that the network resulting from the generated *DL* is not acyclic, a path matrix *PM* is defined and used in the algorithm (Ranjbar et al. [5]). *PM* is an (*m*+*n*+2)(*m*+*n*+2) matrix in which $PM(i, j) = 1$ *iff* there exists a path from node *i* to node *j* and $PM(i, j) = 0$, otherwise.

In each iteration of *BDL*, one element is selected from a restricted candidate list (*RCL*) to be set in *DL*. This element is selected by a biased random procedure. In order to make bias random selection of elements, we use seven rules, defined by Ranjbar et al. [5], and name each of them a priority rule (*pr*). Priority list (*PL*), built in step 2, is a sorting of disjunctive arcs based on priority rule *pr*, see Section 4.3.

In step 3, we determine that whether the algorithm to be reactive or nonreactive by selecting value(s) for α. If α is fixed, we have nonreactive version of the algorithm but if at each iteration, α is selected from a discrete set of possible values, the reactive version of the algorithm is chosen.

Algorithm 2: Building direction list procedure

1: Create *CL* and *PM*
2: Build *PL* based on priority rule *pr*
3: Select α from set Ψ randomly using probability vector **P**
4: *DL*=∅
5: Calculate the incremental penalty cost $Z(e)$ for all $e \in CL$
6: **while** *CL*≠∅ **do**
7: $Z^{\min} = \min\{Z(e) \mid e \in CL\}$
8: $Z^{\max} = \max\{Z(e) \mid e \in CL\}$
9: $RCL = \{e \in CL \mid Z(e) \le Z^{\min} + \alpha(Z^{\max} - Z^{\min})\}$
10: Select an element *e* from the RCL randomly based on vector $\boldsymbol{\pi}$
11: Insert *e* in corresponding position of *DL*
12: Update *PM* and *CL*
13: Recalculate the incremental penalty costs;
14: **end while**
15: Update vector **P**
16: Return *DL*

In the reactive version, the selection of α is guided by the solution values found in the previous iterations. One way to accomplish this is to use the rule proposed by Prais and Ribeiro [22]. Let $\Psi = \{\alpha_1,..., \alpha_k\}$ be the set of possible values for α. In each iteration, α_i has the chance of p_i of being selected given by probability vector $\mathbf{P} = \{p_1,..., p_k\}$ where initially $p_i = 1/k; i = 1,..., k$. Furthermore, let Z^* be the objective function value of the best found solution and let A_i be the average objective function value of all solutions found using $\alpha = \alpha_i; i = 1,..., k$. The selection probabilities are updated (step 15) by taking $p_i = a_i \Big/ \sum_{j=1}^{k} a_j$, with $a_i = Z^*/A_i$ for *i*=1,…,*k*. In step 4, we initialize *DL* as an empty set. In the next step, we calculate the incremental tardiness penalty cost corresponding to all $e \in CL$, shown by Z(*e*). In continue, a while loop is repeated until *CL* is not empty. If $Z^{\min}$ and $Z^{\max}$ show the minimum and maximum of incremental tardiness penalty cost for all $e \in CL$, we define *RCL* as $RCL = \{e \in CL \mid Z(e) \le Z^{\min} + \alpha(Z^{\max} - Z^{\min})\}$.

In step 10, we select an element from *RCL* by a biased random procedure, proposed by Bresina [23]. For this purpose, we rank the elements of *RCL* based on priorities specified by *PL* in which identical rank is considered for both one and zero values of each disjunctive arc. Let $r(e)$ be the rank of $e \in RCL$, we define $\boldsymbol{\pi} = (\pi_1,..., \pi_{|RCL|})$ as the probability vector for selecting *e* from *RCL* in step 10, where π_e is :

$$\pi_e = \frac{1/r(e)}{\sum_{e' \in RCL} (1/r(e'))} \qquad (10)$$

In the next step, selected member element $e \in RCL$ is inserted in corresponding element of *DL*.

In step 11, the *CL* and *PM* are updated as follows. First, we remove from *CL* the element which contains *e* and also the element indicating opposite direction for the disjunctive arc associated to the selected element *e*. Second, if selected element *e* corresponds to arc (i, j), we update *PM* using four following rules:

a) $PM(i, j) = 1$,

b) $PM(i, k) = 1; \forall k \in suc(j)$,

c) $PM(l, i) = 1; \forall l \in pred(i)$

d) $PM(l, k) = 1; \forall l \in pred(i), \forall k \in suc(j)$.

In these four rules, *pred*(*i*) and *suc*(*j*) indicate all (direct and indirect) predecessors and successors of activity *i* respectively, initialized based on set *C* and is updated whenever a new conjunctive arc is added. Rule (a) shows that arc $i \to j$ creates a path between nodes *i* and *j*.

Also, rule (b) indicates that arc $i \to j$ builds a path between node *i* and every node of *suc*(*j*) while rule (c) shows that this new added arc creates a path between every node of *pred*(*i*) and node *j*.

Finally, the last rule demonstrates that arc $i \to j$ builds a path between every node of *pred*(*i*) and every node of *suc*(*j*). Also, for each $\langle i, j \rangle \in D$ that $PM(i, j)$ has been changed after updating, we remove both zero and one elements, corresponding to this disjunctive arc, from *CL* and add the element corresponding to $PM(i, j) = 1$ to *DL*. In step 13, the incremental tardiness penalty costs are recalculated and final *DL* is returned in step 16.

4-2-3. Local Search Procedure

The local search procedure is illustrated by pseudo-code in Algorithm 3. Let G^* be the input graph with direction list DL^*. In the first step, graph G^* which corresponds to the solution $\mathbf{S}^* = \text{CPM}(G^*)$ and objective function Z^* are taken as inputs. Next, we change the value of each element $e_i \in DL^*$ from zero to one or vice versa while other elements are unchanged. This changes the direction of related conjunctive arc and is shown by $inv(e_i)$. In step 4, we update graph G^* and check its feasibility using Floyd-Warshal algorithm (Lawler [24]). If G^* is cyclic, we call repairing procedure (*RP*), shown in Algorithm 4, to make G^* feasible. Of course, the output of *RP* is not always a feasible solution and in this case, we go to the next *i* in step 2. In the repairing procedure, we change values of some $e_j \in DL$ except $j = i$ to remove all loops from G^*. We show the output graph of *RP* by *G*. Also, if changing e_i does not result in a cyclic graph, we only let $G = G^*$ in step 8. This procedure is repeated for all elements of DL^* and whenever an improvement is obtained, the input solution $\mathbf{S}^*$ and its corresponding objective function Z^* are updated. Finally, the best found solution in neighborhood of $\mathbf{S}^*$ or itself is returned as output solution. In the *RP*, the inputs are graph $G^* = (N, C, \sigma^*(D))$ where $\sigma^*(D)$ is specified using DL^* and index *i*. First of all, we check possibility of repairing by letting $C = C \cup arc(e_i)$ where $arc(e_i)$ denotes the directed arc corresponding to e_i. Since the graph in which none of disjunctive arcs are fixed is acyclic, existence of any loop in $G = (N, C)$ implies that no feasible solution can be found while $arc(e_i)$ is included in the project network. In this case, we return "infeasible" as output; otherwise, based on order specified by DL^*, we include directed arcs corresponding to $e_j \in DL^*$ one by one in graph *G*. Whenever a loop is detected, we should include $arc(inv(e_j))$ instead of $arc(e_j)$ in graph G.

Algorithm 3: Local search procedure

1: Let Z^* be the objective function of input solution $\mathbf{S}^*$ where $\mathbf{S}^* = \text{CPM}(G^*)$
2: **for** *i*=1 to $|D|$ **do**
3: $e_i = inv(e_i)$
4: Update G^*
5: **if** G^* is cyclic, **then** $RP(G^*, i)$.
6: **if** $RP(G^*, i)$ is infeasible, **then** go to step 2.
7: **else** $G = RP(G^*, i)$
8: **S**=CPM(*G*)
9: let *Z* as the objective function of graph *G*
10: **if** $Z < Z^*$, **then** ($Z^* = Z$ and $\mathbf{S}^* = \mathbf{S}$)
11: **end for**
12: Return $\mathbf{S}^*$

Algorithm 4: Repairing procedure

1: get G^* and *i* as inputs.
2: Let $C = C \cup arc(e_i)$
3: **if** $G = (N, C)$ is cyclic, **then** go to step 8
4: **for** *j*=1 to $|D|$ and $j \neq i$ **do**
5: $C = C \cup arc(e_j)$
6: **if** $G = (N, C)$ is cyclic, **then** $C = (C \setminus arc(e_j)) \cup arc(inv(e_j))$
7: **end for**
8: Return *G*
9: Return "infeasible".

4-2-4. Path-Relinking Procedure

The idea of our path-relinking procedure, illustrated in Algorithm 5, is taken from Ranjbar et al.2009 [21]. In the first step, we get two direction lists DL and DL' as inputs. In the second step, we assign DL to initial direction list (DL^{in}) and DL' to guiding direction list (DL^{gu}). This assignment is exchanged in step14 and procedure is repeated again. Also, we define child set *CS* as the selected children using *PR* procedure and let it as an empty set in step 2. Next, we let graph set *GS*, a set of generated graphs, as an empty set. In continue, we construct graphs G^{in} and G^{gu} corresponding to direction lists DL^{in} and DL^{gu}. Steps 5 to 12 show a loop in which for *i*=1 to |*D*|, we check whether $e_i^{in} \neq e_i^{gu}$ or $e_i^{in} = e_i^{gu}$. If $e_i^{in} \neq e_i^{gu}$, we change e_i^{in} to $e_i^{in} = e_i^{gu}$ for graph G^{in} in step 8. Next, we check the existence of loop in G^{in}. If it is the case, we apply the *RP* with following changes: remove steps 1,2,3 and 9 from *RP* and consider step 4 for $j = i+1,...,|D|$. This is because G^{gu} is acyclic and $e_j^{in} = e_j^{gu}$ for $j = 1,...,i$, and to make G^{in} an acyclic graph, we need change some values of e_j for $j > i$. In step 10, repaired graph is added to *GS*. At the beginning of step 13, one path of path-relinking has been made. In step 15, one solution is selected from this path and is added to *CS*. The selected solution that is a graph should have direction list different from all members of *ES*. Steps 3 to 13 are repeated by exchanging role of initial and guiding direction lists. After step 14, *CS* has two members and we select the better one using CPM in step 15. Selected member is returned as the output of *PR* procedure in step 16.

Algorithm 5: Path-relinking procedure

1: get DL and DL' as inputs
2: let $DL^{in} = DL$, $DL^{gu} = DL'$ and *CS*=∅
3: Let $GS = \varnothing$
4: Construct graphs G^{in} and G^{gu} corresponding to DL^{in} and DL^{gu}
5: **for** *i*=1 to |*D*| **do**
6: **if** $e_i^{in} \neq e_i^{gu}$, **then**
7: $e_i^{in} = e_i^{gu}$
8: Update G^{in}
9: **if** G^{in} is cyclic, **then** $G^{in} = RP(G^{in}, i)$
10: $GS = GS \bigcup G^{in}$
11: **end if**
12: **end for**
13: select randomly one member from *GS* such that its direction list is different from all members of *ES* and add it to *CS*.
14: Let $DL^{in} = DL'$, $DL^{gu} = DL$, repeat the algorithm one more time from step 3.
15: Find better child solution and let DL'' its corresponding direction list
16: Return DL''

4-3. Priority rules

In this section, we use seven priority rules, developed by Ranjbar et al. [5], to establish the priorities of disjunctive arcs in *PL* where priority values are determined by $\boldsymbol{\lambda} = \{\lambda_{ij}; \langle i, j \rangle \in D\}$. Priority rules are developed based on three characteristics of activities, i.e. precedence relations, durations and resource requirements. In each priority rule, we define a value λ_{ij} for each disjunctive arc $i \leftrightarrow j$ and the sequence of arcs in *PL* is made by non-increasing order of λ_{ij} values. As a tie breaker, λ_{ij} with smaller *i* and then smaller *j* gets priority. Table 2, taken from Ranjbar et al. [5], shows the formula of each priority rule and the contributing characteristics. In the priority rule 1, only precedence relations of activities are contributing. In this rule, λ_{ij} equals the summation of the total number of successors of activities *i* and *j*. Similar to priority rule 1, in priority rules 2 and 3 only one characteristic is contributing. In priority rule 2, λ_{ij} equals the summation of the durations of activities *i* and *j* while in priority rule 3 it equals the summation of θ_r values for all $r \in R_i$ or R_j where $\theta_r = w_r / (\delta_r - \rho_r)$ and R_i denotes the set of required resources for execution of activity $i \in N$.

Each one of priority rules 4, 5 and 6 is based on the contribution of two characteristics. In priority rule 4, the precedence relations and durations are contributing, the summation of tails of activities *i* and *j* is considered as λ_{ij}. Tail of activity $i \in N$, shown by q_i, is a lower bound for the time period between the completion of activity *i* and the project deadline and is calculated using equation (11).

$$q_i = \max\{d_j + q_j ; (i,j) \in C\} \quad (11)$$

Equation (11) requires initialization which is given by $q_{n+1} = 0$.

Two characteristics, precedence relations and resource requirements are contributing in rule 5 in which λ_{ij} equals the summation of θ_r values for all $r \in R_k$ where *k* is representative of all activities belonging to at least one of the *pred*(*i*), *pred*(*j*), *suc*(*i*) *or suc*(*j*) and requiring at least one common resource with activity *i* and *j*. Rule 6 is based on the combination of two characteristics, durations and resource requirements, while in the last priority rule all three characteristics are contributed.

Tab. 2. Priority rules 1 to 7

Rule number	λ_{ij}	Characteristics		
		Precedence relations	durations	Resource requirements
1	$\lvert suc(i) \cup suc(j) \rvert$	✓		
2	$d_i + d_j$		✓	
3	$\sum_{r \in (R_i \cup R_j)} \theta_r$			✓
4	$q_i + q_j$	✓	✓	
5	$\sum_{k \in (pred(i) \cup pred(j) \cup suc(i) \cup suc(j)),\ R_k \cap (R_i \cup R_j) \neq \varnothing} \ \sum_{r \in R_k} \theta_r$	✓		✓
6	$d_i \sum_{r \in R_i} \theta_r + d_j \sum_{r \in R_j} \theta_r$		✓	✓
7	$d_i \sum_{r \in R_i} \theta_r + d_j \sum_{r \in R_j} \theta_r + \sum_{k \in (pred(i) \cup pred(j) \cup suc(i) \cup suc(j)),\ R_k \cap (R_i \cup R_j) \neq \varnothing} d_k \sum_{r \in R_k} \theta_r$	✓	✓	✓

Table 3 illustrates the result of application of each priority rule on the example project. In this table, set $\boldsymbol{\lambda}$ and its corresponding priorities list (*PL*) of the disjunctive arcs are shown for each rule.

Tab. 3. Results of application of priority rules 1 to 7 on the example project

Rule number	$\boldsymbol{\lambda} = \{\lambda_{12}, \lambda_{14}, \lambda_{25}, \lambda_{34}, \lambda_{35}, \lambda_{45}, \lambda_{56}\}$	*PL*
1	{5,4,4,2,2,2,1}	$\{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle, \langle 5,6\rangle\}$
2	{5,9,7,10,9,11,8}	$\{\langle 4,5\rangle, \langle 3,4\rangle, \langle 1,4\rangle, \langle 3,5\rangle, \langle 5,6\rangle, \langle 2,5\rangle, \langle 1,2\rangle\}$
3	{0.176,0.4,0.4,0.4,0.4,0.4,0.4}	$\{\langle 1,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle, \langle 5,6\rangle, \langle 1,2\rangle\}$
4	{16,10,9,6,3,3,0}	$\{\langle 1,2\rangle, \langle 1,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle, \langle 5,6\rangle\}$
5	{0.4,0.22,0.22,0.22,0.22,0.22,0.4}	$\{\langle 1,2\rangle, \langle 5,6\rangle, \langle 1,4\rangle, \langle 2,5\rangle, \langle 3,4\rangle, \langle 3,5\rangle, \langle 4,5\rangle\}$
6	{0.8,2.9,2.3,3.2,2.8,4.3,2.6}	$\{\langle 4,5\rangle, \langle 3,4\rangle, \langle 1,4\rangle, \langle 3,5\rangle, \langle 5,6\rangle, \langle 2,5\rangle, \langle 1,2\rangle\}$
7	{5.2,6.8,6.8,4.8,4.4,5.9,6.8}	$\{\langle 1,4\rangle, \langle 2,5\rangle, \langle 5,6\rangle, \langle 4,5\rangle, \langle 1,2\rangle, \langle 3,4\rangle, \langle 3,5\rangle\}$

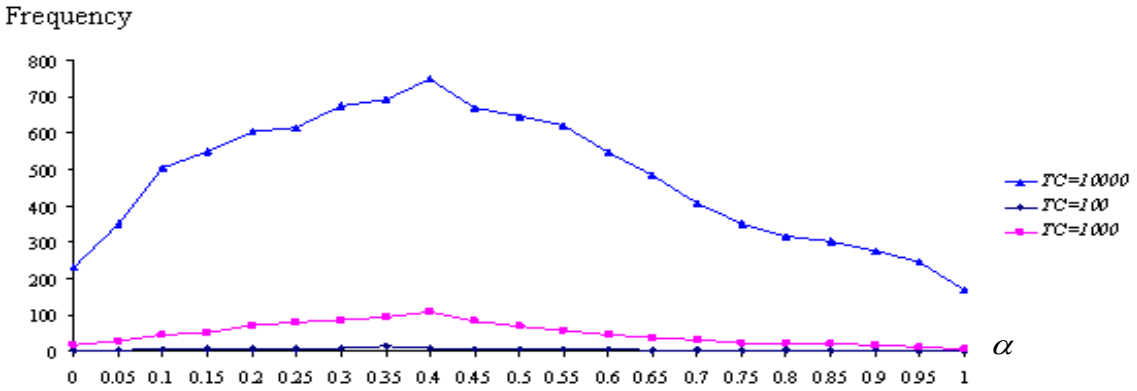

Fig. 2. Frequency of different values of α

5. Computational Experiments

5-1. Benchmark Problem Sets

We have coded the procedure in Visual C++6 and performed all computational experiments on a PC Pentium IV 3GHz processor with 1024 MB of internal memory. In order to evaluate the performance of our algorithm, we used test problems generated by Ranjbar et al. [5].

The test problems are generated for full factorial of three parameters, i.e. the number of activities (*n*), the network shape parameter, order strength[2] (*OS*), and the resource factor[3] (*RF*).

They consider five values 20, 22, 24, 26 and 28 for *n*, three values 0.2, 0.35 and 0.5 for *OS* and three values 0.1, 0.2 and 0.3 for *RF*. For each combination of *n*, *OS* and *RF*, they generate three test instances giving rise to 135 test instances. We also set the number of resource to *m*=3.

Also, for each resource *r*, we select ρ_r, δ_r and w_r randomly from discrete uniform distributions $U[1,n]$, $\rho_r + U\left[0.8\times\sum_{i\in N_r} d_i, 1.2\times\sum_{i\in N_r} d_i\right]$ and $U[1,m]$ respectively.

We run our algorithm for three values of *TC* as *TC*=100, 1000 and 10000.

5-2. Parameter Setting

One of the benefits of GRASP is that it has smaller number of parameters than other metaheuristics. Since we have used reactive version of GRASP, the parameter α is set automatically.

For this purpose, we consider set Ψ as Ψ ={0, 0.05, 0.1, …, 1} in which parameter α is changed in a range between zero and one with step size of 0.05. The case α=0 corresponds to a pure greedy algorithm, while α=1 is equivalent to a random construction. Figure 2 shows the frequency of different values of α used in *TC* iterations when priority rule 4 (the best priority rule) is used. All three curves have a bell-shape in which maximum frequency for *TC*=100 occurs for α=0.35 while for *TC*=1000 and 100000 occur for α=0.4.

5-3. Comparative Computational Results

In this section, we first compare the results of the algorithm obtained based on different priority rules. Next, a comparison between reactive and nonreactive versions of the algorithm is done. In continue, we investigate the impact of local search and path-relinking procedures. Comparison criterion is average percent deviation (APD) from optimal solutions, obtained by Ranjbar et al. [5].

5-3-1. Impact of Priority Rules

Table 4 shows the APD for solutions using different priority rules and three termination criteria. The results show a consistent ranking of priority rules for different values of *TC*.

This rank is 4, 2, 1, 5, 7, 6 and 3. Priority rule 4 that can be considered as a combination of priority rules 1 and 2 has the smallest *APD*. After that, priority rules 2 and 1 have the second and third smallest *APD*, respectively.

Priority rule 3, based on resource requirements, has largest *APD*.

It can be concluded that priority rules in which durations and precedence relations of activities are contributing have better results than priority rules in which resource requirements of activities are contributing.

The average CPU run time for *TC*=100, 1000 and 10000 are 0.05, 0.63 and 6.84 seconds.

It is interesting to report that this ranking is consistent with the results reported by Ranjbar et al. [5].

[2] The order strength is the number of comparable intermediate activity pairs divided by the maximum number $n(n\text{-}1)/2$ of such pairs, and is a measure for the closeness to a linear order of the technological precedence constraints in *C* (cfr. Mastor, 1970).

[3] The resource factor shows how many numbers of resources are used in average by each of the activities.

Tab. 4. *APD* for different priority rules and termination criteria

	TC		
pr	**100**	**1000**	**10000**
1	65.2	30.3	19.0
2	64.6	27.3	15.7
3	92.1	58.6	36.4
4	61.7	25.4	12.4
5	73.0	41.8	24.1
6	81.9	53.3	31.3
7	75.4	45.1	29.3

5-3-2. Comparison of Reactive and Nonreactive Versions of the Algorithm

In this section, we compare two versions of our algorithm, reactive and nonreactive. For this comparison, we consider only priority rule 4, the best priority rule, and set parameter α for different values of *TC* based on fine tuning.

For nonreactive version, we set α to 0.35 when *TC*=100 and set it to 0.4 when *TC*=1000 and 10000. The results of nonreactive version of the algorithm in which α has a fixed value are shown in Table 5. Also, we have shown in this table the results of the cases α=0 and 1.

Tab. 5. *APD* for different values of α and termination criteria

	TC		
α	**100**	**1000**	**10000**
0.00	90.6	61.6	52.3
0.35	78.9	-	-
0.40	-	38.7	20.3
1.00	107.8	79.5	68.7

The results show that the reactive version of algorithm outperforms the nonreactive version. When we consider *TC*=100 iterations, *APD* in nonreactive version is 78.9 while this value in reactive version is 61.7, shown in Table 4. Also, when we set *TC* to 1000 and 10000 iterations, *APD*s in nonreactive version are 38.7 and 20.3 while corresponding values in reactive version are 25.4 and 12.4, respectively. Furthermore, pure greedy algorithm (the case α=0) and random algorithm (the case α=1) give rise to worse results than other cases. If we compare the results of the first and the last rows of Table 5, we see that pure greedy algorithm is better than random algorithm.

5-3-3. Impact of Local Search Procedure

Table 6 shows the results obtained from running the algorithm without local search procedure in which reactive version is considered.

Tab. 6. *APD* for the algorithm without local search procedure

	TC		
pr	**100**	**1000**	**10000**
1	84.7	40.1	25.9
2	81.0	36.8	25.4
3	105.4	63.8	47.6
4	75.8	35.3	21.1
5	87.3	48.5	32.7
6	97.3	55.5	36.9
7	92.6	52.9	41.2

If we compare the results of tables 4 and 6, we surely conclude that for all priority rules and termination criteria, local search has improved *APD*. Of course, when we exclude the local search procedure, the CPU run times are a bit smaller than the case in which local search in included. The new average CPU run times corresponding to *TC*=100, 1000 and 10000 are 0.04, 0.39 and 4.51 seconds.

5-3-4. Impact of Path-Relinking Procedure

In order to evaluate the impact of path-relinking procedure, we exclude it from the algorithm and obtained new results, shown in Table 7.

Tab. 7. *APD* for the algorithm without path-relinking procedure

	TC		
pr	**100**	**1000**	**10000**
1	69.3	31.9	22.2
2	68.3	28.8	20.2
3	95.4	59.5	39.6
4	62.5	28.0	18.3
5	76.7	43.5	28.9
6	84.6	54.7	33.9
7	79.5	47.7	33.4

Similar to previous section, it is seen that for all priority rules and termination criteria the results are worse than the results of Table 4. Of course, it should be noticed that the CPU run times are decreased when *PR* is excluded from the algorithm. The new average CPU run times corresponding to *TC*=100, 1000 and 10000 are 0.04, 0.35, 3.87 seconds.

6. Summary and Conclusions

In this paper, we presented the problem of minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling. We developed a metaheuristic algorithm, based on GRASP and path-relinking, accompanied with a local search procedure. We considered two reactive and nonreactive versions of algorithm and showed, using computational experiments, that reactive version of the algorithm outperforms nonreactive version. Also, we used seven priority rules to bias the random selection of elements from *RCL*. These priority rules are defined based on three characteristics of activities: precedence relations, durations and resource requirements. The computations experiments showed the best results are for the priority rule defined based on the combination of two characteristics, i.e. durations and precedence relations of activities. Moreover, we demonstrated the improving role of local search and path-relinking procedures using computational experiments.

An important research direction that might be pursued in the future is extension of developed priority rules in this work. Also, developing other metaheuristic algorithms for problem defined in this paper is an interesting research topic.

References

[1] Blazewicz, J., Lenstra, J., Rinnooy-Kan, A.H.G., *Scheduling Subject to Resource Constraints-Classification and Complexity*, Discrete Applied Mathematics 5, 1983, pp. 11- 24.

[2] Demeulemeester, E., Herroelen, W., *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers 2002.

[3] Neumann, K., Schwindt, C., Zimmermann, J., *Project Scheduling with Time Windows and Scarce Resources*, Springer 2002.

[4] Herroelen, W., *Project Scheduling-Theory and Practice*, Production and Operations Management 14, 2005, pp. 413-432.

[5] Ranjbar, M., Khalilzadeh, M., Kinafar, F., Etminani, K., *An Optimal Procedure for Minimizing Total Weighted Resource Tardiness Penalty Costs in the Resource-Constrained Project Scheduling Problem*, Computers and Industrial Engineering 62, 2012, pp. 264-270.

[6] Khalilzadeh, M., Kianfar, F., Shirzadeh Chaleshtari, A., Shadrokh, S., Ranjbar, M., *A Modified PSO Algorithm for Minimizing the Total Costs of Resources in MRCPSP*, Mathematical Problems in Engineering, 1, 2012, pp.1-18.

[7] Vanhoucke, M., Demeulemeester, E., Herroelen, W., *An Exact Procedure for the Resource-Constrained Weighted Earliness-Tardiness Project Scheduling Problem*, Annals of Operations Research 102, 2001, pp. 79-196.

[8] Nadjafi, B.A., Shadrokh, S., *A Branch and Bound Algorithm for the Weighted Earliness-Tardiness Project Scheduling Problem with Generalized Precedence Relations*, Scientia Iranica, 16, 2009, pp. 55-64.

[9] Bellman, R.E., *On a Routing Problem*, Quarterly Applied Mathematics, 16, 1985, pp.87-90.

[10] Aiex, R.M., Resende, M.G.C., Ribeiro, C.C., *Probability Distribution of Solution Time in GRASP: an Experimental Investigation*, Journal of Heuristics, 8, 2002, pp. 343-373.

[11] Feo, T.A., Resende, M.G.C., *Greedy Randomized Adaptive Search Procedures*, Journal of Global Optimization, 6, 1995, pp. 109-133.

[12] Feo, T.A., Resende, M.G.C., Smith, S., *A Greedy Randomized Adaptive Search Procedure for Maximum Independent Se*t, Operations Research, 42, 1994, pp. 860-878.

[13] Hart, J.P., Shogan, A.W., *Semi-Greedy Heuristics: an Empirical Study*, Operations Research Letters, 6, 1987, pp. 107-114.

[14] Glover, F., *Tabu Search and Adaptive Memory Programming—Advances*, applications and challenges. In: R.S. Barr, R.V. Helgason and J.L. Kennington (eds.), Interfaces in Computer Science and Operations Research. Kluwer, 1996, pp. 1–75.

[15] Glover, F., Laguna, M., *Tabu Search*, Kluwer Academic Publishers, 1997.

[16] Glover, F., Laguna, M., Marti, R., *Fundamentals of Scatter Search and Path Relinking*, Control and Cybernetics, 39, 2000, pp. 653–684.

[17] Laguna, M., Marti, R., *GRASP and Path Relinking for 2-Layer Straight ;Line Crossing Minimization*, INFORMS Journal on Computing, 11, 1999, pp. 44–52.

[18] Ribeiro, C.C., Uchoa, E., Werneck, R.F., *A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs*, INFORMS Journal on Computing, 14, 2002, pp. 228–246.

[19] Resennde, M.G.C., Marti, R., Gallego, M., Duarte, A., *GRASP and Path-Relinking for the Max-Min Diversity Problem*, Computers & Operations Research, 37, 2010, pp. 498-508.

[20] Alvarez-Valdes, R., Crespo, E., Tamarit, J.M., Villa, E., *GRASP and Path-Relinking for Project Scheduling under Partially Renewable Resources*, European Journal of Operational Research, 189, 2008, pp.1153-1170.

[21] Ranjbar, M., De Reyck, B., Kianfar, F., *A Hybrid Scatter Search for the Discrete Time/Resource Trade-off Problem in Project Scheduling*, European Journal of Operational Research, 193, 2009, pp. 35-48.

[22] Prais, M., Ribeiro, C.C., *Reactive GRASP: an Application to a Matrix Decomposition Problem in TDMA Traffic Assignment*, INFORMS Journal on Computing, 12 2000, pp. 164–176.

[23] Bresina, J.L., *Heuristic-Biased Stochastic Sampling, In: Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, 1996, pp. 271–278.

[24] Lawler, E.L., *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York 1976.