Contents lists available at SciVerse ScienceDirect

Computers & Industrial Engineering

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

An optimal procedure for minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling problem ☆

Mohammad Ranjbar [a,*], Mohammad Khalilzadeh [b], Fereydoon Kianfar [b], Kobra Etminani [c]

[a] Department of Industrial Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, P.O. Box 91775-1111, Mashhad, Iran
[b] Department of Industrial Engineering, Sharif University of Technology, Tehran, Iran
[c] Department of Computer Engineering, Faculty of Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

ARTICLE INFO

Article history:
Received 7 September 2010
Received in revised form 9 May 2011
Accepted 20 September 2011
Available online 25 September 2011

Keywords:
Resource-constrained project scheduling
Weighted tardiness
Branch-and-bound

ABSTRACT

We present an optimal solution procedure for minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling problem. In this problem, we assume the constrained renewable resources are limited to very expensive equipments and machines that are used in other projects and are not available in all periods of time of a project. In other words, for each resource, there is a dictated ready date as well as a due date such that no resource can be available before its ready date but the resources are permitted to be used after their due dates by paying penalty cost depending on the resource type. We also assume that only one unit of each resource type is available and no activity needs more than it for execution. The objective is to determine a schedule with minimal total weighted resource tardiness penalty costs. For this purpose, we present a branch-and-bound algorithm in which the branching scheme starts from a graph representing a set of conjunctions (the classical finish-start precedence constraints) and disjunctions (introduced by the resource constraints). In the search tree, each node is branched to two child nodes based on the two opposite directions of each undirected arc of disjunctions. Selection sequence of undirected arcs in the search tree affects the performance of the algorithm. Hence, we developed different rules for this issue and compare the performance of the algorithm under these rules using a randomly generated benchmark problem set.

1. Introduction

The resource-constrained project scheduling problem (RCPSP) involves the non-preemptive scheduling of project activities subject to finish-start-type precedence constraints and renewable resource constraints in order to minimize the project duration. It is shown in Blazewicz, Lenstra, and Rinnooy-Kan (1983) that the RCPSP, as a job-shop generalization, is NP-hard in the strong sense. A large number of exact and heuristic procedures have been proposed to construct workable baseline schedules that solve this problem; see Demeulemeester and Herroelen (2002), Kolisch and Padman (2001) and Neumann et al. (2002) for recent overviews and Herroelen (2005) for a discussion on the link between theory and practice. In addition, Kolisch and Hartmann (2006) present a classification and performance evaluation of different heuristic and metaheuristic algorithms for the RCPSP. The RCPSP under minimization of total weighted resource tardiness penalty cost (RCPSP–TWRTPC) is an applicable problem and a modified version of the RCPSP in which all assumptions and constraints of the RCPSP are held but the objective is different. We assume that the renewable resources such as labor are not limited and expensive but there are a few renewable resources like especial types of crane, tunnel boring machines, very expert human resources that should be hired outside of the project. Since these limited renewable resources are used in other projects, there is a dictated ready date as well as a due date for each of them such that no resource can be available before its ready date but these resources are permitted to be used after their due dates by paying penalty cost, depending on the resource type. We also assume there is at most one unit of each limited resource type in each period of time of the project horizon and no activity needs more than one unit for execution while it may need more than one resource type. Bianco, Dell'Olmo, and Speranza (1998) referred to the resources that can be assigned to only one activity at a time as dedicated resources. The goal of problem is to find a schedule with minimal total weighted resource tardiness. The RCPSP–TWRTPC can also be considered as a modified version of the job-shop scheduling problem with minimization of the total weighted tardiness in which the tardiness is proposed for jobs and technical precedence relations are only among operations of a job. This problem is known to be strongly NP-hard since

☆ This manuscript was processed by Area Editor Willy Herroelen.
* Corresponding author. Tel./fax: +98 511 8796778.
E-mail addresses: m_ranjbar@um.ac.ir (M. Ranjbar), khalilzadeh@mehr.sharif.edu (M. Khalilzadeh), kianfar@sharif.edu (F. Kianfar), etminani@wali.um.ac.ir (K. Etminani).

0360-8352/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2011.09.013

the single machine problem $1\|\sum w_jT_j$ had been proved to be strongly NP-hard by Lenstra, Rinnooy, and Brucker (1977).

To the best of our knowledge, we are introducing the RCPSP–TWRTPC for the first time and there is no previous work in the literature. The only related problems proposed in the literature are scheduling problems 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. Some of the proposed exact algorithms are as follows: Vanhoucke, Demeulemeester, and Herroelen (2001) 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 (2009) developed a B&B algorithm for the weighted earliness–tardiness project scheduling problem with generalized precedence relations. Singer and Pinedo (1998) presented a computational study of two B&B techniques for minimizing the total weighted tardiness in job shops. They compared the computational results of two branching schemes, i.e. the operation insertion branching scheme and the arc insertion branching scheme. Pan, Chen, and Chao (2002) studied two-machine flow shop problem with goal of minimizing tardiness and developed a B&B algorithm for the problem. Also, Shim and Kim (2007) proposed a B&B algorithm accompanied with efficient dominance properties and lower bounds for scheduling on parallel identical machines to minimize total tardiness. Furthermore, Liaw, Lin, Cheng, and Chen (2003) developed a B&B algorithm for scheduling unrelated parallel machines to minimize total weighted tardiness. In the context of single machine problems, Tian, Ng, and Cheng (2005) addressed single machine total tardiness problem in which a number of distinct due date are given. They identified some optimal properties and proved that the problem is polynomially solvable.

Several heuristic algorithms are proposed for machine scheduling problems with objective function related to the tardiness. Essafi, Yazid, and Dauzère-Pérès (2008) presented a genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Bilge, Kiraç, Kurtulan, and Pekgün (2004) developed a tabu search algorithm for parallel machine total tardiness problem. Also, Bilge, Kurtulan, and Kiraç (2007) proposed a tabu search algorithm for the single machine total weighted tardiness problem.

The contributions of this article are fourfold: (1) we introduce and formulate the RCPSP_TRWPTC by a disjunctive graph; (2) we design a very fast heuristic procedure to develop an initial solution; (3) we develop a B&B algorithm based on arc insertion for the proposed problem; (4) we develop seven rules for the sequence of arc insertion and show, using computational experiments, that the rules defined on the basis of duration and precedence relation characteristics make the algorithm more efficient than other rules.

The remainder of this article is organized as follows. Problem modeling and formulation are provided in Section 2. Section 3 presents our branch-and-bound algorithm. The computational experiments are presented in Section 4. Finally, summary and conclusions are given in Section 5.

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 $N = \{0, 1, \ldots, n + 1\}$ 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 $C = \{(i, j);\ i \to j, i, j \in N\}$ consists of arcs representing technical finish-to-start precedence constraints among activities, where $i \to j$ implies activity j can be started after finishing of activity i. Let $R = \{1, 2, \ldots, m\}$ the set for constrained renewable resources and N_r the set of activities which need one unit of resource $r \in R$ for execution. If there exists a resource $rinR$ for which two activities $i, j \in N_r$, there is a disjunctive arc $i \to j$ between nodes i and j. Thus we present the set of disjunctive arcs as $D = \{\langle i,j \rangle;\ i \leftrightarrow j, i < j, \exists r \in R : i,j \in N_r\}$. Since availability of each resource is at most one unit in each period of time, two activities i and j, where $\langle i,j \rangle \in D$ cannot 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 release dates in the graph representation, we add one node corresponding to each resource to the project network. For 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 $i \in N_r$. 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, $m = 2$ resources and the corresponding graph is depicted in Fig. 1. 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 resource 1 and 2, respectively. Precedence relations of each of these nodes with dummy node 0 and its successors (the nodes which require these resources) 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 a vector $\mathbf{S} = (s_1, s_2, \ldots, s_n)$, where s_i is integer and shows the start time of activity i. Given a policy for scheduling, such as earliest 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 critical path (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 $f_i = s_i + d_i$ of activity i is equal to $L(0, i)$, computed using the algorithm of Bellman(1958) 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 computed as $T_r = \max\{c_r - \delta_r, 0\}$. The total weighted resource tardiness penalty cost is $\sum_{r=1}^{m} w_rT_r$.

Table 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}

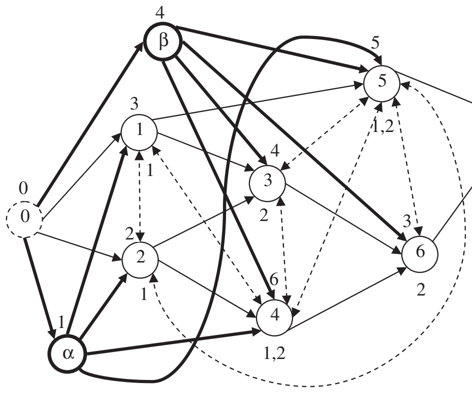

Fig. 1. Graph of the example project.

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

$$\min Z = \sum_{r=1}^{m} w_r T_r \tag{1}$$

Subject to :

$$c_r \geqslant s_i + d_i \text{ for } r = 1,\dots,m; \quad i \in N_r \tag{2}$$
$$T_r \geqslant c_r - \delta_r \text{ for } r = 1,\dots,m \tag{3}$$
$$T_r \geqslant 0 \text{ for } r = 1,\dots,m \tag{4}$$
$$s_i \geqslant \rho_r \text{ for } r = 1,\dots,m; \quad i \in N_r \tag{5}$$
$$s_j - s_i \geqslant d_i \text{ for all } (i,j) \in C \tag{6}$$
$$s_j - s_i \geqslant d_i - M(1 - X_{ij}) \text{ for all } \langle i,j\rangle \in D \tag{7}$$
$$s_i - s_j \geqslant d_j - MX_{ij} \text{ for all } \langle i,j\rangle \in D \tag{8}$$
$$s_i, c_r, T_r \in N^+ \text{ for } i = 0,1,\dots,n+1; \quad r = 1,2,\dots,m \text{ and } X_{ij} \in \{0,1\} \text{ for all } \langle i,j\rangle \in D \tag{9}$$

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 each activity which requires that resource. Constraint sets (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 release 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 in which M denotes a very big positive number. Finally, constraint (9) ensure that variables s_i, c_r and T_r are non-negative integers and X_{ij} is a binary variable.

3. A branch-and-bound algorithm

In this section we develop a depth-first B&B algorithm for the RCPSP_TRWTPC. In this algorithm we first develop a heuristic procedure to generate an initial solution, giving an upper bound (*UB*) for the beginning of the B&B. Whenever a better new solution is obtained, the *UB* will be updated. Next, the branching, based on the arc insertion scheme, is started. In order to prevent the generation of infeasible or dominated solutions, we develop some efficient bounding rules. At the end of this section, we develop different rules for the sequence of arc insertion and then show how this sequence may affect the efficiency of the algorithm.

3.1. A heuristic procedure for an initial solution

In this procedure, resources are selected one by one and for each selected resource r, a direction is fixed for each disjunctive arc $\langle i,j\rangle$, where $i, j \in N_r$. In order to implement this procedure, we first sort the resources based on a criterion that is a combination of penalty costs, ready dates and due dates. For this purpose, for each resource $r \in R$, we define $\pi_r = \frac{w_r}{\delta_r - \rho_r}$ and sort the resources based on the non-increasing order of π_r values. Intuitively, the resources with higher weight of penalty costs and tighter availability intervals $(\delta_r - \rho_r)$ have higher priorities than others to be selected. Then, starting from highest value of π_r, for each resource r, we generate a priority list PL_r for all activities $i \in N_r$. In order to create PL_r, we first define tail q_i for each activity $i \in N$ as a lower bound for the time period between the completion of activity i and the project deadline. A procedure for calculating tails is presented in Brucker, Jurisch, and Kramer (1994). This procedure has time complexity $O|C|$ and calculates tails using

$$q_i = \max\{d_j + q_j; (i,j) \in C\} \tag{10}$$

Eq. (10) requires initialization which is given by $q_{n+1} = 0$. The priority list PL_r is constructed based on the non-increasing order of q_i values for $i \in N_r$. As a tie breaker for both π_r and q_i, we sort them based on the decreasing order of their indexes.

If the obtained PL_r does not satisfy the precedence constraints imposed by conjunctive arcs, it is called infeasible. If it is the case, we next generate the precedence feasible list PL_r that is a modified version of PL_r. The PFL_r is generated from the PL_r using a procedure including $|N_r|$ iterations, where $|N_r|$ is the size of N_r. At the beginning of this procedure, we consider a sub-network consisting of all activities $i \in N_r$. If two activities i and j belong to this sub-network and there is at least a path from i to j in the main network, we consider activity i as one of the predecessors of activity j in the sub-network. We define the set of eligible activities E to be all activities in the corresponding sub-network that do not have any predecessor. The activity $i^* \in E$ having the highest q-value is the first member of the PFL_r. Then, we remove i^* from the corresponding sub-network and update the set E. Now, for every pair of activities i^* and $j \in N$ that $\langle i^*,j\rangle \in D$, we change the disjunctive arc $i^* \leftrightarrow j$ to the conjunctive arc $i^* \to j$. This is repeated until the PFL_r includes all activities $i \in N_r$, i.e. |N_r| iterations.

The procedure of making *PFL* is applied to all $r \in R$ on the order of non-increasing π_r. To prevent cycle creation, whenever the procedure is applied for certain r, the precedence relations on the main network are updated and precedence feasible lists for remaining resources are obtained using the updated precedence relations.

For the example project, we have $\pi_1 = 0.176$ and $\pi_2 = 0.222$. Since $\pi_2 > \pi_1$, we first select resource 2 for which $N_2 = \{3, 4, 5, 6\}$. The values of tails for activities 1 to 6 are as follows: $q_1 = 7, q_2 = 9, q_3 = 3, q_4 = 3, q_5 = 0$ and $q_6 = 0$. For creation of PFL_2, we should consider the sub-network including activities 3,4,5 and 6. In this sub-network, activities 3 and 4 are the predecessors of activities 6. Thus, at the beginning, we have $E = \{3,4,5\}$. Since $q_3 = q_4 > q_5$, we consider activity 3, having smaller index, as the first member of the PFL_2. As a result, we change the disjunctive arcs $3 \leftrightarrow 4$ and $3 \leftrightarrow 5$ to the conjunctive arcs $3 \to 4$ and $3 \to 5$ respectively. Now, we update set E as $E = \{4,5\}$ and select activity 4 as the second member of the PFL_2 in the next iteration. The outcome of this selection is converting the disjunctive arc $4 \leftrightarrow 5$ to the conjunctive arc $4 \to 5$. In the next iteration, activity 5 is selected and the conjunctive arc $5 \to 6$ is added. By applying this procedure on resource 1, the disjunctive arcs $1 \leftrightarrow 2$, $1 \leftrightarrow 4$ and $2 \leftrightarrow 5$ will be converted to the conjunctive arcs $2 \to 1$, $1 \to 4$ and $2 \to 5$ respectively. The release time of resources will be $c_1 = 21$ and $c_2 = 24$. Consequently, the weighted resource tardiness penalty cost of the obtained solution equals 17.

3.2. Branching scheme

Our branching strategy is based on the arc insertion scheme in which each node in the branching tree represents a selection of disjunctive arcs $\sigma(D^P)$, where D^P is a subset of D. $\sigma(D^P)$ has a conjunctive arc for every pair in D^P and the resulting graph $G = (N, C, \sigma(D^P))$ is acyclic. Two child nodes are generated by branching on a pair $i \leftrightarrow j$ that is not in D^P; one node for $i \to j$ and another node for $j \to i$. Thus, in the worst case, for a problem with $|D|$ conjunctive arcs, our algorithm has $2^{|D|}$ nodes.

3.3. A cycle detection procedure

The insertion of an arc introduces precedence relationships between two activities using a common resource. Suppose the arc $i \to j$ is in a selection $\sigma(D^P)$ and a new arc $j \to k$ is inserted. Then the arc $i \to k$ will be redundant while the arc $k \to i$ cannot be selected because it would create a cycle. For instance, if the arc $1 \to 2$ is in a selection $\sigma(D^P)$ of the example project, due to the existence of arc $2 \to 4$, then the arc $1 \to 4$ will be redundant while the arc $4 \to 1$ creates the cycle $1 \to 2 \to 4 \to 1$.

In order to detect whether there is any cycle in each node of the search tree, we first define a path matrix $PM_{(n+m+2)(n+m+2)}$ in which the $PM(i, j) = 1$ if there is a path from node i to node j in the graph $G = (N, C)$ and $PM(i,j) = 0$ otherwise. Since the graph $G = (N,C)$, corresponding to the root node of the search tree, is acyclic, we monitor and prohibit the cycle creation in each node of the search tree using following procedure.

For every two activities $\langle i,j\rangle \in D$, we can change disjunctive arc $i \leftrightarrow j$ to arc $i \to j$ if $PM(j,i) = 0$. Obviously, if $PM(j,i) = 1$, arc $i \to j$ creates a cycle. Now, if $PM(j,i) = 0$ and we introduce arc $i \to j$, we update PM using the following four rules: (a) $PM(i,j) = 1$, (b) $PM(i,k) = 1;\ \forall k \in suc(j)$, (c) $PM(l,j) = 1;\ \forall l \in pred(i)$, and (d) $PM(l,k) = 1;\ \forall l \in pred(i),\ \forall k \in suc(j)$. In this four rules, $pred(i)$ and $suc(i)$ indicate all (direct and indirect) predecessors and successors of activity i respectively, initialized based on the set C and may be 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, by the last rule, arc $i \to j$ builds a path between every node of $pred(i)$ and every node of $suc(j)$. In the worst case, the time order of our proposed procedure is $O(n^2)$, where PM is updated by rule (d).

Of course, instead of our developed method for the cycle detection, we could remove the unselected disjunctive arcs and check the existence of any cycle using the Floyd–Warshall algorithm (see e.g. Lawler (1976)) in each nodes of the search tree. Obviously, our procedure is more efficient because the time order of the Floyd–Warshall algorithm is $O(n^3)$ (Lawler (1976)). However, In each node of the search tree, if a cycle is detected, that node should be fathomed.

3.4. A lower bound

In this section, we develop a lower bound for the optimal solution based on the idea introduced by Stinson, Davis, and Khumawala (1978) in which a lower bound is computed for the project makespan by considering both precedence and resource constraints. We construct the lower bound of the optimal solution by building a lower bound for the release time of each resource $r \in R$. Each node of the search tree corresponds to a network, named the current network, and includes a number of disjunctive and conjunctive arcs. If we consider only the conjunctive arcs of the current network and we calculate the earliest starting time (es_i) of each activity $i \in N_r$ using the critical path method (CPM), the lower bound for c_r, shown by lb_r, can be obtained by

$$lb_r = \max_{i\in N_r}\{es_i + d_i + \sum_{j\in B_i} d_j\} \tag{11}$$

In this formula, B_i includes activities $j \in N_r$ for which $es_j \geqslant es_i$. To justify Eq. (11), we note that the activities of set N_r needs at least one common resource and cannot be executed in parallel. The lower bound for the optimal solution (LB) can be easily obtained by $LB = w_r \max\{lb_r - \delta_r, 0\}$. In each node of the search tree, if the corresponding LB is more than the UB, that node should be fathomed.

3.5. Sequence of the arc insertion

For each selection $\sigma(D)$ of the disjunctive arcs with feasible schedule, the sequence of the arc insertion does not affect the feasibility or the solution quality but it may affect the efficiency of the algorithm. In the search tree of B&B algorithm, we may find many infeasible or dominated solutions. Thus, if we detect these solutions earlier at higher levels of the search tree, we prevent the tree from expanding and find optimal solution quicker. Since generation of infeasible or dominated solutions are common in arc insertion, we have tested different rules for the sequence of arc insertion to reduce the total CPU run time of the algorithm.

In the following we develop seven rules to determine the sequence of arc insertion. These rules are developed based on three characteristics of activities, i.e. precedence relations, duration, and resource requirements. In each rule, we define a value w_r for each disjunctive arc δ_r. The sequence of arc insertion is based on the non-increasing order of ρ_r values. As the first tie breaker, we sort disjunctive arcs based on non-decreasing order of first index of elements. As the second tie breaker, disjunctive arcs are sorted based on increasing order of second index of elements.

Table 2 shows different rules and the contributing characteristics. In rule 1, only precedence relations of activities are contributing. In this rule, λ_{ij} equals total number of successors of activities i and j. Similar to rule 1, in rules 2 and 3 only one characteristic is contributing. In rule 2, λ_{ij} equals the summation of durations of activities i and j while in the rule 3 it equals the summation of π_r values for all $r \in R_i$ or R_j, where R_i indicates the set of required resources for execution of activity i. Each of the rules 4–6 is based on the contribution of two characteristics. In rule 4, precedence relations and durations are contributing, the summation of tails of activities i and j is considered as λ_{ij}.

Two characteristics, precedence relations and resource requirements, are contributing in rule 5 in which λ_{ij} equals the summation of π_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 having at least a common required resource with R_i or R_j. Rule 6 is based on the combination of two characteristics, durations and resource requirements, while in the last rule all three characteristics are contributing.

Table 3 illustrates the result of application of each rule on the example project. This table shows λ_{ij} for each disjunctive arc $i \leftrightarrow j$ of the example project and the corresponding sequence of the disjunctive arcs.

4. Computational experiments

4.1. Benchmark problem sets

We coded our algorithm in Java and performed all computational experiments on a PC Pentium IV 3 GHz processor with 1024 MB of internal memory. We run the B&B algorithm with each of the seven rules developed in the previous section. The B&B using rule i; is referred to $i = 1, 2, \ldots, 7$ as B&B(i).

In order to evaluate the performance of B&B(i) for $i = 1, 2, \ldots, 7$, we generated test problems using the random network generator RanGen (Demeulemeester, Vanhoucke, & Herroelen, 2003). The test problems are generated for full factorial of three parameters, i.e. the number of activities (n), the network shape parameter *order strength*[1] (OS), and the resource factor[2] (RF). We considered 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. There are two reasons for considering small values for RF. Considering higher values for RF increases the computation time unreasonably while comparative

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

[2] The resource factor shows how many number of resources are used, on the average, by each of the activities.

Table 2
Rules 1–7 for the sequence of the arc insertion.

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

Table 3
Results of application of rules 1–7 on the example project.

Rule number	$\{\lambda_{12}, \lambda_{14}, \lambda_{25}, \lambda_{34}, \lambda_{35}, \lambda_{45}, \lambda_{56}\}$	Sequence
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}	$\{\langle 1,4\rangle, \langle 2,5\rangle, \langle 5,6\rangle, \langle 4,5\rangle, \langle 1,2\rangle, \langle 3,4\rangle, \langle 3,5\rangle\}$

results show the outcomes very similar to what presented in this paper. We found this on the basis of considering values 0.25, 0.5 and 0.75 for *RF* and testing on a small set of instances. On the other hand, in practice, the activities that need very expensive resources are not really more than (almost) 30° of all activities of a project.

For each combination of *n*, *OS* and *RF*, we generated 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 selected ρ_r and w_r randomly from discrete uniform distributions $U[1, n]$ and $U[1, m]$, respectively. In addition, for each resource *r*, we set δ_r to $\rho_r + cp$, where *cp* indicates the length of critical path.

The average number of disjunctive arcs for different values of *n*, *OS* and *RF* is presented in Table 4. As it is expected, the number of disjunctive arcs is increased by increasing n and RF and decreasing OS.

4.2. Comparative Computational results

In this section, total CPU time for running a B&B algorithm is referred to as T_{Total} and the CPU time spent for finding optimal solution is referred to as T_{Best}. The most important factor affecting T_{Total} and T_{Best} is the number of disjunctive arcs, presented in Table 4. T_{Total} and T_{Best}, expressed in seconds, are displayed in Table 5. Since T_{Total} for some of the B&B algorithms were very large, we limited the T_{Total} to 1 h (3600 s). Thus, for the cases that T_{Total} is greater than 3600 s, we consider them equal to 3600 s. The last row of Table 5 indicates how many number of optimal solutions, shown by *No. Opt*, are found in 1 h. If we look at Tables 2 and 5 together, we can conclude that durations and precedence relations are more important characteristics than resource requirements in determination of the arc insertion sequence. This conclusion is compatible when we combine two characteristics durations and precedence relations and obtain the best results, shown by B&B(4). If we compare the results of B&B(4) and B&B(7), we see that contribution of resource requirements characteristic in finding a rule for the sequence of arc insertion makes the results worse. Similar results are obtained if we compare B&B(1) with B&B(5) and B&B(2) with B&B(6).

In the following, we interpret the reason of efficiency ranking of the B&B algorithms. The efficiency of our developed B&B algorithms depends on the number of redundant disjunctive arcs and fathomed nodes. There are two general rules for cutting nodes in our B&B algorithms; cycle detection and lower bound. In the project network, each disjunctive arc includes two nodes; begin node and end node. In terms of cycle detection and redundant arcs, wherever the number of input conjunctive arcs to the begin node of a disjunctive arc increases or the number of output conjunctive arcs from the end node of a disjunctive arc increases, the probability of detecting cycle or finding redundant arcs will be raised. On the other hand, cutting nodes based on the LB depends on both activities duration and resource characteristics but it seems the activities duration are more important.

Table 4
Average number of disjunctive arcs.

OS	RF	*n*				
		20	22	24	26	28
0.2	0.1	3.7	4	6	6.7	5.3
	0.2	14.7	21	25	36	28.7
	0.3	32	36.7	50.3	67	73.7
0.35	0.1	2.3	3.3	4	6.3	7
	0.2	14.7	14.7	18	21.7	31
	0.3	24.3	32.7	46	47.3	60
0.5	0.1	3.3	3.3	2.3	3	3.3
	0.2	11	11	12.3	16.3	16.7
	0.3	18.3	22.3	30.7	34.3	49.3

Table 5
Summary results of the B&B algorithms.

	B&B(1)	B&B(2)	B&B(3)	B&B(4)	B&B(5)	B&B(6)	B&B(7)
Avg. T_{Best}	1.68	0.95	8.62	0.51	3.68	8.41	20.12
Avg. T_{Total}	20.86	4.14	75.98	3.81	20.70	57.28	30.87
No.Opt	135	135	133	135	135	133	134

Table 6
Average T_{Total} in seconds for different values of n.

n	20	22	24	26	28
B&B(1)	0.02	0.03	1.23	4.38	98.63
B&B(2)	0.01	0.03	0.66	3.03	16.96
B&B(3)	0.07	0.06	14.33	10.25	355.21
B&B(4)	0.01	0.03	0.92	2.54	15.53
B&B(5)	0.04	0.05	2.88	18.12	82.39
B&B(6)	0.05	0.05	3.71	18.09	264.5
B&B(7)	0.05	0.06	2.67	2.81	148.76

Table 7
Average T_{Total} in seconds for different values of OS.

OS	0.2	0.3	0.4
B&B(1)	61.06	1.05	0.45
B&B(2)	10.48	1.09	0.85
B&B(3)	160.57	62.87	4.51
B&B(4)	10.27	0.71	0.44
B&B(5)	36.96	23.87	1.25
B&B(6)	91.09	79.27	1.48
B&B(7)	94.76	4.97	1.88

Table 8
Average T_{Total} in seconds for different values of RF.

RF	0.1	0.2	0.3
B&B(1)	0	0.05	62.53
B&B(2)	0	0.04	12.37
B&B(3)	0	0.22	227.73
B&B(4)	0	0.03	11.39
B&B(5)	0	0.06	62.03
B&B(6)	0	0.08	171.76
B&B(7)	0	0.06	92.58

Table 9
Performance of B&B algorithms for limited CPU run times.

CPU run time	1	5	10	30	60
B&B(1)	5.55(125)	2.49(130)	1.07(132)	0.63(133)	0.51(134)
B&B(2)	5.51(121)	1.28(129)	1.02(131)	0.11(134)	0.00(135)
B&B(3)	13.41(116)	11.09(120)	8.98(125)	5.16(126)	3.08(129)
B&B(4)	3.27(126)	0.48(133)	0.12(133)	0.04(134)	0.00(135)
B&B(5)	6.86(121)	4.66(126)	3.11(129)	0.66(132)	0.39(133)
B&B(6)	12.83(116)	8.02(121)	7.05(122)	1.92(128)	1.53(131)
B&B(7)	12.18(121)	4.95(125)	1.52(129)	0.89(131)	0.64(133)

Tables 6–8 display T_{Total} obtained by each of B&B algorithms for different values of n, OS and RF respectively. As it is expected, T_{Total} is increased by increasing n and RF and by decreasing OS. This behavior is consistent and does not depend on the type of B&B algorithm.

In order to compare the performance of B&B algorithms for short run times, we run each of the B&B algorithms for 1, 5, 10, 30 and 60 s. The results are shown in Table 9 in which each cell includes two values, average percent deviation from optimal solution (*APD*) and *No. Opt*. Again, we see that B&B(4) outperforms other B&B algorithms while B&B(2) and B&B(1) are the next best algorithms, respectively.

4.3. Performance of the heuristic procedure

In this section, we evaluate the performance of the heuristic procedure developed to generate the initial solution. For this purpose, we computed the *APD* of the initial solutions from optimal solutions for different values of n, OS and RF, shown in Tables 10–12 respectively. Table 10 implies that there is no relation between *APD* and number of activities while Tables 11 and 12 show a trend. Table 11 indicates the APD is decreased by increasing of the order strength while Table 12 indicates the APD is increased by increasing of the resource factor.

Table 10
Average percent deviation from optimal solution for different values of n.

n	20	22	24	26	28
APD	108.45	154.15	74.92	79.61	83.37

Table 11
Average percent deviation from optimal solution for different values of OS.

OS	0.2	0.35	0.5
APD	135.49	132.97	31.84

Table 12
Average percent deviation from optimal solution for different values of RF.

RF	0.1	0.2	0.3
APD	85.64	105.11	109.55

5. Summary and conclusions

In this paper, we studied the problem of minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling. We showed the problem as a graph including conjunctive and disjunctive arcs and we formulated it as an integer programming model. As the solution approach, we developed a first-depth branch-and-bound algorithm, based on the arc insertion technique, for this problem. We also developed a fast heuristic procedure for generation of initial solution and a lower bound to accelerate the algorithm. A complete solution is obtained when for each disjunctive arc, a direction is selected. Although the sequence of arc insertion does not have any effect on the value of optimal solution, but it affects the speed of the algorithm. If infeasible or dominated solutions are found in the higher level of the search tree, they are discarded to decreases the generated nodes and run time of algorithm. We developed seven rules for the sequence of arc insertion defined based on the three characteristics of activities: precedence relations, durations and resource requirements. The computations experiments show that the best results are related to the rules defined based on the combination of two characteristics, i.e. durations and precedence relations of activities.

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

References

Bellman, R. E. (1958). On a routing problem. *Quarterly Applied Mathematics, 16*, 87–90.

Bianco, L., Dell'Olmo, P., & Speranza, M. G. (1998). Heuristics for multimode scheduling problems with dedicated resources. *European Journal of Operational Research, 107*, 260–271.

Bilge, Ü., Kiraç, F., Kurtulan, M., & Pekgün, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers & Operations Research, 31*, 397–414.

Bilge, Ü., Kurtulan, M., & Kiraç, F. (2007). A tabu search algorithm for the single machine total weighted tardiness problem. *European Journal of Operational Research, 176*, 1423–1435.

Blazewicz, J., Lenstra, J., & Rinnooy-Kan, A. (1983). Scheduling subject to resource constraints – Classification and complexity. *Discrete Applied Mathematics, 5*, 11–24.

Brucker, P., Jurisch, B., & Kramer, A. (1994). The job-shop problem and immediate selection. *Annals of Operations Research, 50*, 73–114.

Demeulemeester, E., Herroelen, W., 2002. *Project scheduling: A research handbook*. Kluwer Academic Publishers.

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

Essafi, I., Yazid, M., & Dauzère-Pérès, S. (2008). A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research, 35*, 2599–2616.

Herroelen, W. (2005). Project scheduling-theory and practice. *Production and Operations Management, 14*, 413–432.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research, 174*, 23–37.

Kolisch, R., & Padman, R. (2001). An integrated survey of deterministic project scheduling. *Omega, 29*, 249–272.

Lawler, E. L. (1976). *Combinatorial optimization: Networks and matroids*. New York: Holt, Rinehart and Winston.

Lenstra, J., Rinnooy, A. K., & Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics, 1*, 343–362.

Liaw, C., Lin, Y., Cheng, C., & Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers & Operations Research, 30*, 1777–1789.

Mastor, A. A. (1970). An experimental and comparative evaluation of production line balancing techniques. *Management Science, 16*, 728–746.

Nadjafi, B. A., & Shadrokh, S. (2009). A branch and bound algorithm for the weighted earliness–tardiness project scheduling problem with generalized precedence relations. *Scientia Iranica, 16*, 55–64.

Neumann, K., Schwindt, C., & Zimmermann, J. (2002). Recent results on resource constrained project scheduling with time windows: Models, solution methods, and applications. *Central European Journal of Operations Research, 10*, 113–148.

Pan, J. C., Chen, J., & Chao, C. (2002). Minimizing tardiness in two-machine flow shop. *Computers & Operations Research, 29*, 869–885.

Singer, M., & Pinedo, M. (1998). A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions, 30*, 109–118.

Shim, S. O., & Kim, Y. D. (2007). Scheduling on parallel identical machines to minimize total tardiness. *European Journal of Operational Research, 177*, 135–146.

Stinson, J. P., Davis, E. W., & Khumawala, B. M. (1978). Multiple resource-constrained scheduling using branch and bound. *AIIE Transactions, 10*, 252–259.

Tian, Z. J., Ng, C. T., & Cheng, T. C. E. (2005). On the single machine total tardiness problem. *European Journal of Operational Research, 165*, 843–846.

Vanhoucke, M., Demeulemeester, E., & Herroelen, W. (2001). An exact procedure for the resource-constrained weighted earliness–tardiness project scheduling problem. *Annals of Operations Research, 102*, 179–196.