Hindawi
Advances in Operations Research
Volume 2017, Article ID 1308704, 11 pages
https://doi.org/10.1155/2017/1308704

Research Article

Resource Tardiness Weighted Cost Minimization in Project Scheduling

Ali Shirzadeh Chaleshtari

Department of Industrial Engineering, College of Engineering, Islamic Azad University, Tehran Gharb Branch, Tehran, Iran

Correspondence should be addressed to Ali Shirzadeh Chaleshtari; shirzadeh.a@wtiau.ac.ir

Received 10 August 2016; Revised 30 November 2016; Accepted 4 December 2016; Published 10 January 2017

Academic Editor: Yi-Kuei Lin

In this paper, we study a project scheduling problem that is called resource constrained project scheduling problem under minimization of total weighted resource tardiness penalty cost (RCPSP-TWRTPC). In this problem, the project is subject to renewable resources, each renewable resource is available for limited time periods during the project life cycle, and keeping the resource for each extra period results in some tardiness penalty cost. We introduce a branch and bound algorithm to solve the problem exactly and use several bounding, fathoming, and dominance rules in our algorithm to shorten the enumeration process. We point out parameters affecting the RCPSP-TWRTPC degree of difficulty, generate extensive sets of sample instances for the problem, and perform comprehensive experimental analysis using the customized algorithm and also CPLEX solver. We analyze the algorithm behavior with respect to the changes in instances degree of difficulty and compare its performance for different cases with the CPLEX solver. The results reveal algorithm efficiency.

1. Introduction

Resource constrained project scheduling problems are widely studied in the open literature. Various criteria are studied in these problems. Project makespan minimization is one of the most studied objectives which is aimed in the resource constrained project scheduling problem (RCPSP). Resource tardiness minimization is another objective that has been much less studied in the literature. This objective is studied in the current paper. It differs from makespan minimization, since it tries to schedule those activities earlier for which the due dates of the required resources are earlier, even though this may elongate the project makespan. The optimal schedule in this case can change by simply changing the due dates of the resources. But in makespan minimization, the due dates of the resources are not taken into account.

Resource tardiness minimization is a pervasive objective in the practical cases of the project scheduling. In many projects, resources are available for limited periods of the project life cycle and keeping them available after the related periods is subject to some tardiness costs. Renewable resources that are rented from outside the project or the ones that are common between various projects are usually subject to this condition. For example, in construction projects, heavy machines such as tower cranes or bulldozers are usually subject to this condition. These resources may also have ready times; that is, they may not be ready at the beginning of the project. Resource tardiness cost minimization is a preferred objective in scheduling of these projects.

In this paper we study an extension of the RCPSP problem that is called resource constrained project scheduling problem under minimization of total weighted resource tardiness penalty cost (RCPSP-TWRTPC) [1]. In this problem the project is subject to renewable resources. Each renewable resource is available for limited time periods during the project life cycle and keeping the resource for each extra period results in some tardiness penalty cost. The objective is to minimize the total resource tardiness penalty cost. We introduce a branch and bound algorithm to solve the problem exactly. The branching structure in this algorithm is similar

to the precedence tree approach, which is a basic approach to solve the RCPSP problem [2]. We use several bounding, fathoming, and dominance rules in our algorithm to shorten the enumeration process. The algorithm performance is numerically analyzed. The results reveal algorithm efficiency.

The rest of this paper is organized as follows. In the next section, the related literature is reviewed; in Section 3 the RCPSP-TWRTPC problem and in Section 4 the branch and bound algorithm to solve the problem are described in detail. Section 5 is dedicated to the experimental analysis and finally, Section 6 contains a few concluding remarks.

2. Related Work

The literature on the RCPSP problem dates back to 1960s [3]. Different studies have developed exact algorithms for the problem, such as the binary programming based model of Patterson and Roth [4], the dynamic programming algorithm of Carruthers and Battersby [3], and the branch and bound methods of Brucker et al. [5], Yoosefzadeh and Tareghian [6], and Dorndorf et al. [7]. Due to the fact that the problem is NP-hard [8, 9], many inexact methods have been published for RCPSP as well, such as the heuristic methods of Ying et al. [10], Zamani [11], Kolisch [12], and Ranjbar [13] and the metaheuristic methods like Valls et al. [14] which are based on genetic algorithm, Shukla et al. [15] and Bouleimen and Lecocq [16] which are based on simulated annealing, Thomas and Salhi [17] and Nonobe and Ibaraki [18] which are based on tabu search, [19] which is based on particle swarm optimization, and Agarwal et al. [20] which is based on an artificial immune system based approach. Many review papers have summarized the problem literature, such as Herroelen et al. [21], Hartmann and Kolisch [22], Kolisch and Padman [23], and Kolisch and Hartmann [24]. Finally, the problem has been extended in different ways, such as the extensions to multiple objectives [25], multiple activity modes [26], time-cost tradeoff [27], time-cost-quality tradeoff [28], stochastic activity duration time [29], and multiple-projects [30, 31]. Some other extended forms of the problem are the hybrids of the aforementioned extended forms, such as stochastic time-cost tradeoff problem studied in Ke et al. [32] and Ke et al. [33].

Many project scheduling problems with resource-oriented objectives have been defined and studied in the literature. Some of these problems are *resource allocation problem* studied in Azaron and Tavakkoli-Moghaddam [34], *resource investment problem* studied in Najafi and Niaki [35], and *resource availability cost problem* studied in Ranjbar et al. [36]. A survey of variants of these studies can be found in Hartmann and Briskorn [37] and Węglarz et al. [38].

Resource tardiness cost minimization has been studied in few papers. Ranjbar et al. [1] introduced RCPSP-TWRTPC and proposed a branch and bound algorithm for the problem. However the problem they study differs from the problem we study here, as in their problem only unary renewable resources are considered; that is, the maximum availability of the resources in each period and also the maximum resource requirement of each activity are one. In Ranjbar [39] a hybrid grasp algorithm was proposed for this same problem. In Khalilzadeh et al. [40] an extended form of RCPSP-TWRTPC was introduced and studied. In that problem which is called *multimode RCPSP-TWRTPC (MRCPSP-TWRTPCP)*, project activities are multimode and they are subject to nonrenewable resources in addition to the renewable resources. The problem aims at the minimization of the total cost of the resources, including the renewable resources tardiness weighted cost and the cost of usage of nonrenewable resources by project activities. To solve this problem, a metaheuristic algorithm was introduced based on a modified version of particle swarm optimization method.

3. Problem Description

A project with n nondummy activities is considered. There exist finish-start precedence relations between activities which are illustrated using an activity-on-node (AON) loopless network, with dummy nodes 1 and $n + 1$ as initial and terminal nodes, respectively. Let $K = \{1, \ldots, \mathrm{NR}\}$ be the set of renewable resources. Each activity j ($j = 1, \ldots, n + 1$) has a fixed duration d_j and requires r_{jk} units of renewable resource k ($k \in K$) for each unit of time over its duration. Besides, activity j has a set of predecessor activities, P_j. Every renewable resource k has constant availability of R_k for each period over the project duration, the ready time rt_k, the deadline dt_k, and the unit tardiness penalty cost of pc_k; that is, the resource is not available before the period rt_k, and after the period dt_k for each period of the resource tardiness, a penalty cost of pc_k results. No preemption is permitted during the execution of the activities, all of the activities have single mode, and they are ready at the beginning of the project horizon. All parameters are integers. The problem is to find the start time of each activity j, S_j, ($j = 1, \ldots, n + 1$), such that all problem constraints are satisfied and the total cost of renewable resource tardiness is minimized.

Suppose that the earliest start time of each activity j, EST_j, ($j = 1, \ldots, n+1$), is determined with forward pass. In order to get more adequate EST for each activity, we minimally limit the EST of activities using the renewable resource constraints. Based on this, the earliest start time of each activity cannot be earlier than the earliest time that all renewable resources that are required by the activity are ready. We also determine the latest start time of each activity j, LST_j, ($j = 1, \ldots, n + 1$), using backward pass. In order to do that, we specify an upper bound T on the project makespan of every optimum solution for the problem and fix the latest start time of the last dummy activity equal to this upper bound. We determine T by summing the durations of the entire activities and adding the latest ready time of the renewable resources to this sum. It can be easily shown that T is an upper bound on the makespan of at least one optimal solution for the problem.

We now define the following decision variables:

$$x_{jt} = \begin{cases} 1 & \text{if activity } j \text{ starts in period } t \\ 0 & \text{otherwise} \end{cases}$$

$$j = 1, \ldots, n + 1, \; t = \mathrm{EST}_j, \ldots, \mathrm{LST}_j.$$

$$y_{kt} = \begin{cases} 1 & \text{if renewable resource } k \text{ is used in period } t \\ 0 & \text{otherwise} \end{cases} \quad k \in K,\ t = 0, \ldots, T; \tag{1}$$

l_k is renewable resource k tardiness.

Then we have, $S_j = \sum_{t=\mathrm{EST}_j}^{\mathrm{LST}_j} t \cdot x_{jt}$ and the mathematical model of the problem is as the following:

$$\text{Min} \quad \sum_{k=1}^{\mathrm{NR}} \mathrm{pc}_k \cdot l_k \tag{2}$$

$$\sum_{t=\mathrm{EST}_j}^{\mathrm{LST}_j} x_{jt} = 1, \quad j = 1, \ldots, n+1 \tag{3}$$

$$\sum_{t=\mathrm{EST}_i}^{\mathrm{LST}_i} (t + d_i) \cdot x_{it} \le \sum_{t=\mathrm{EST}_j}^{\mathrm{LST}_j} t \cdot x_{jt}, \quad j = 1, \ldots, n+1,\ i \in P_j \tag{4}$$

$$\sum_{j=1}^{n} \sum_{\tau=\max(t-d_j+1, \mathrm{EST}_j)}^{\min(t, \mathrm{LST}_j)} r_{jk} x_{j\tau} \le R_k \cdot y_{kt}, \quad k \in K,\ t = 0, \ldots, \mathrm{LST}_n \tag{5}$$

$$\sum_{k=1}^{\mathrm{NR}} \sum_{t=0}^{rt_k - 1} y_{kt} = 0 \tag{6}$$

$$t \cdot y_{kt} - dt_k \le l_k, \quad k \in K,\ t = dt_k, \ldots, \mathrm{LST}_n \tag{7}$$

$$x_{jt} \in \{0, 1\}, \quad j = 1, \ldots, n+1,\ t = \mathrm{EST}_j, \ldots, \mathrm{LST}_j \tag{8}$$

$$y_{kt} \in \{0, 1\}, \quad k \in K,\ t = 0, \ldots, \mathrm{LST}_{n+1} \tag{9}$$

$$l_k \ge 0, \quad k \in K. \tag{10}$$

In the model above, objective (2) is the minimization of the project scheduling cost. Constraints (3) guarantee that each activity j can only have a single start time from the period $[\mathrm{EST}_j, \mathrm{LST}_j]$. Constraints (4) take into consideration precedence relations between each pair of activities (i, j) where i is an immediate predecessor of j. Constraints (5) regard renewable resource usage limitation, according to constraints (6) renewable resources may be used after their ready times, constraints (7) determine the tardiness value of each renewable resource, and finally, constraints (8), (9), and (10) denote the domain of the variables.

It can be shown that the RCPSP-TWRTPC problem is an extension of RCPSP and therefore, it is NP-hard.

4. Branch and Bound Algorithm for RCPSP-TWRTPC

In this section we describe our branch and bound algorithm for solving the RCPSP-TWRTPC. This method is based on partial schedules in which only parts of the project activities are scheduled. In each node of the branch and bound tree, one activity is selected and scheduled until all activities are scheduled. Several bounding, fathoming, and dominance rules are used in the algorithm to shorten this enumeration process.

Basic scheme of the branch and bound algorithm for solving RCPSP-TWRTPC, describing each part of the algorithm in detail, is as follows:

(1) Perform preprocessing and stop if the instance is infeasible

(2) Determine disjunctive pairs of activities

(3) Specify the initial upper bound on the optimal objective function value

(4) Generate the initial node and select it for branching

(5) Branch the selected node

(6) Check every new node according to the dominance rules and fathom the checked node if necessary

(7) Close each new node containing feasible solution and update the current upper bound

(8) If there is at least one open node yet:

  (a) Select a new node for branching according to the selection method

  (b) Perform fathoming checks on the selected node and continue from step (5) if the node is not fathomed; otherwise continue from step (8)

Else:

  (a) Report the best feasible solution achieved and stop

4.1. Preprocessing. There is only one case in which a given instance of RCPSP-TWRTPC has no feasible solution. In this case, there is shortage for at least one of the renewable resources; that is, there exists an activity like j and a renewable resource like k in the problem for which $r_{jk} > R_k$. In this situation the activity cannot be executed and no feasible solution exists for the problem. Our algorithm checks the feasibility of the instance in the first step.

4.2. Disjunctive Pairs of Activities. Two activities are in disjunction if resources available in the problem are not enough for both of them to execute concurrently, so one of these disjunctive activities has to finish before the other one can start. For a pair of disjunctive activities, the following test can be applied to possibly introduce some precedence relations between them:

Interval based disjunctive consistency test, Carlier and Pinson [41]: for two disjunctive activities of i and j, if $\mathrm{LST}_i\text{-}\mathrm{EST}_j < d_j$, i must precede j.

We determine disjunctive activities at the beginning of our algorithm to perform this test during the fathoming check at every node of the branching tree. If some precedence relation already exists between a pair of activities, the test brings no new information to the solution method. So we only determine disjunctive pairs without any existing precedence relations between the related activities.

4.3. Initial Upper Bound Specification. In order to determine an initial upper bound on the optimum objective function value, we generate a feasible solution for the problem using the following heuristic method and use its related objective function value as the initial upper bound. In this heuristic method, first, an *activity list* (AL) is generated which shows the priority of the activities in generating the related schedule; that is, the ith element in the list has the priority of i in the schedule generation process. In order to generate a precedence feasible schedule, we generate a precedence feasible activity list in which each activity has a lower priority than its predecessors. In order to generate this list, we use a quantity related to each activity which we call *maximum resource tardiness cost of unit extra period* (MTC). MTC for each activity is sum of the unit tardiness costs of those renewable resources that the activity requires. In order to generate the AL, activities are sorted in nonincreasing order of their MTC considering the precedence relations.

In the second step after the generation of the AL, activities are scheduled using the *serial schedule generation scheme* (SSGS). The schedule generated in this way is a feasible and usually good solution for the problem.

4.4. Initial Node Generation. The branching tree of the algorithm is initiated with the initial node. The related schedule in this node contains the first dummy activity which is scheduled in the first period. This node is generated and then branched as the only available node according to specific structure of the branch and bound algorithm.

4.5. Branching. In our algorithm, branching is performed similar to the precedence tree algorithm for the RCPSP problem. Regarding each selected node g for branching, the set of *eligible activities* (EJ_g) for this node is determined that contains all activities not scheduled yet whose predecessors have already been scheduled. For each member of this set like j_g, a new node is generated and j_g is scheduled in the earliest feasible time considering the precedence and resource constraints. In appendix of this paper, we show that at least one optimum schedule is gained among the schedules generated in this way.

4.6. Dominance Rules. Several dominance rules have been introduced for the precedence tree algorithm for solving the RCPSP. Some of them have been reviewed in Demeulemeester and Herroelen [42]. According to these dominance rules, some of the nodes in the branching tree may be fathomed before branching them to shorten the enumeration process because better or the same quality nodes exist in the branching tree.

Two of the dominance rules of the precedence tree algorithm for the RCPSP are applicable in our algorithm here. Experimental analyses show that these rules are very effective in the algorithm efficiency. Description of these rules is as follows.

Regarding each partial schedule in each node, we can specify an activity list that contains the list of activities in the order of their start times. When some activities have the same start times, it is possible to relate more than one activity list to a given partial schedule. In this case, there can be one node associated with each of these activity lists in the branching tree. Therefore if we have two nodes with different activity lists whose related partial schedules are the same, we can fathom one of them, because they are related to the similar solutions. Noting to this, if in some node, for a selected activity j and a previously scheduled activity i we have $S_j < S_i$, we fathom the generated node, because it contains the same partial schedule as another node whose activity list is the same but orders of activities i and j are reverse in the list. Another case is when activity j is scheduled in the node with the same start time as the already scheduled activity i. In this case the node contains the same partial schedule with another one in which the orders of i and j are substituted. So we can fathom one of these nodes and as a rule, in such cases we fathom new node if $j < i$.

4.7. Closing Feasible Nodes and Upper Bound Update. After each branching process, each newly generated node like g whose related schedule contains the entire project activities is closed. Then the related objective function value of the schedule, say F_g, is compared with the current upper bound and the upper bound is updated as F_g if it is more than F_g.

4.8. Node Selection for Branching. We use depth-first-search method to keep the memory requirements of the algorithm low. According to this method, an open node is selected from the higher level of the branching tree. If a tie happens, three rules are used in respective order until one single node is chosen, including selecting a node with the most scheduled activities, a node with the least associated lower bound, and the node with the least index number.

4.9. Fathoming Check. Fathoming check is performed on each open node g selected for branching. The effectiveness of this test depends on the current value of the upper bound; that is, tighter upper bound values lead to more effectiveness in the test. Since the upper bound may be improved during the algorithm, we delay the fathoming check of each unfathomed node as much as possible. So the check is performed when the node is selected for branching.

Pseudocode of fathoming check of each selected node for branching, showing the main steps of this process, is as follows:

(1) Determine EST of the unscheduled activities

(2) Determine the related lower bound of the node

(3) Fathom the node if the related lower bound of the node is not less than the current upper bound and go to step (8)

(4) Determine LST of the activities

(5) Fathom the node if EST of an unscheduled activity is more than its LST and go to step (8)

(6) Perform interval based disjunctive consistency test and make the new relations effective

(7) Continue from step (5) if the node is not fathomed and EST of any unscheduled activity has been modified

(8) End the fathoming process

4.9.1. Determining EST of Unscheduled Activities. EST of each unscheduled activity is determined based on the precedence relations using forward pass. However, the forward pass process is modified here by making three modifications. Firstly, the EST of every scheduled activity is fixed equal to its start time, secondly, a minimum on the EST of each unscheduled activity is determined equal to the earliest time that all renewable resources required by the activity are ready, and thirdly, a minimum on the start time of each unscheduled activity is determined using the two dominance rules that were described in Section 4.6.

4.9.2. Determining the Related Lower Bound of the Node. A lower bound is determined on the objective function values of all the nodes that may be generated via branching of the selected node g. In order to do that, start time of every unscheduled activity is supposed equal to its EST; then the lower bound is specified equal to the objective function value of the resulting schedule.

4.9.3. Determining LST of the Activities. The LST of each scheduled activity is considered equal to its start time and an upper bound on the start time of every unscheduled activity i, LST_i, is determined using the current upper bound. In order to do that, it is supposed that the related partial schedule of the node g is completed by scheduling the activity i in a period S_i and every other unscheduled activity in the earliest feasible period considering the precedence relations. Then LST_i is specified as the latest possible value for S_i in a way that the related objective function value of the completed schedule remains less than the current upper bound.

Based on this definition for the LST of an unscheduled activity, we perform the following three main steps to determine the latest start time of every unscheduled activity. In these steps, we note that the LST of an activity cannot be more than its LST in its parent node. We also use some sets of activities that we define as the following. For every renewable resource k in the problem, we define *the set of the last project activities requiring the renewable resource* k (SLA_k) as the set of activities that require the resource k for execution, but none of their successors requires this resource. Based on this definition, the last period that the resource k is used during the project life cycle equals the last period of execution of the activities of SLA_k.

Step 1. For every unscheduled activity i and every renewable resource k, we determine the latest period that the activity i may be scheduled in the current partial schedule so that the current tardiness of the resource k does not change. We call this period RLST_{ik} and determine it for every resource k and every unscheduled activity i in decreasing order of activities numbers as the following:

(i) If $i \in \mathrm{SLA}_k$, RLST_{ik} equals the latest period that the resource k is required in the current partial schedule of the node minus $(d_i + 1)$.

(ii) If the activity i and none of its successors belong to SLA_k, RLST_{ik} may be as large as possible. In this case we set RLST_{ik} equal to an upper bound for the project makespan of the optimal solution.

(iii) If the activity i does not belong to SLA_k, but at least one of its successors does, then RLST_{ik} is determined using backward pass based on RLST_{jk} of all other activities $j > i$.

Step 2. In this step, we first set the LST of every unscheduled activity equal to its EST. Then we increase its LST one unit iteratively and in each iteration, we determine the related increase in the tardiness cost of the resources. For each value of the LST for each unscheduled activity i, the increase in the tardiness of a resource k equals the maximum of zero and LST_i-RLST_{ik}. So the LST can be increased as much that the total increase in the tardiness cost plus the current lower bound of the node remains less than the current upper bound of the problem.

Step 3. In this step the feasibility of the latest start time of the activities that were determined in the previous step is checked based on the precedence relations. The backward pass is performed and if necessary, the latest start times are lessened and corrected.

4.9.4. Performing the Interval Based Disjunctive Consistency Test. After the determination of the EST and LST of the activities in the node, we perform the interval based disjunctive consistency test for each pair of the disjunctive activities. We note that the possible relations introduced by this test in each node g are only specific to the node g and its further branches.

Having performed the test in the node g, if some new relations are introduced between the activities, we made them effective by updating the EST and LST of the activities. Let us suppose that a new relation makes an activity i the immediate predecessor of the activity j in a node g. This new relation is made effective in the following way:

(i) If the activity j is an unscheduled activity in the node, either the activity i is scheduled or not:

(a) If $\mathrm{EST}_i + d_i > \mathrm{EST}_j$, EST_j is updated as $\mathrm{EST}_i + d_i$ and if necessary, EST of every successor of the activity j is updated using forward pass.

(b) If EST of an activity increases here, the related lower bound of the node should be updated too and if it gets more than the current upper bound of the problem, the node is to be fathomed.

(ii) If the activity j and not the activity i is scheduled:

(a) If $\mathrm{LST}_i + d_i > \mathrm{LST}_j$, LST_i is updated as $\mathrm{LST}_j - d_i$ and if necessary, LST of every predecessor of the activity i is updated using backward pass. In this case, if LST of the activity i or one of its predecessors turns less than the EST of the related activity, the node is fathomed.

5. Experimental Analysis

In this part we present the results of a comprehensive computational experiment that we conducted regarding the algorithm presented in this paper for the RCPSP-TWRTPC. The algorithm was coded and executed on C#.NET 2010 platform. In addition, the CPLEX solver version 12.4 was used in the analyses. Experimental tests were performed on a PC with Core 2 Duo 2.53 GHz CPU and 3 GBs RAM.

In the following sections, we first describe the sample problems that we used and then we present and discuss the results categorized in four parts. We used the customized algorithm and also the CPLEX solver to solve each sample problem. A time limit of 10 seconds was imposed on the execution time for solving each instance by the algorithm and by the CPLEX solver, so that the experiments could be completed within a reasonable time. In order to evaluate the effectiveness of the algorithm for solving the RCPSP-TWRTPC, we used two metrics. The first metric is the number of instances solved within 10 seconds and the second metric is the actual execution time for solving every instance that was solved within 10 seconds.

5.1. Sample Problems. As the RCPSP-TWRTPC is an extension of the RCPSP, all parameters that affect the degree of difficulty of an instance of the RCPSP (i.e., the computational requirement of any algorithm for solving the instance) are likely to affect the corresponding degree of difficulty of an instance of the RCPSP-TWRTPC, too. Besides, the additional parameters regarding the renewable resources may affect the degree of difficulty of the RCPSP-TWRTPC instances. We distinguished two parameters in this regard which we describe in the following. The actual effects of these parameters in the degree of difficulty of the instances are experimentally viewed in the computational results in the next sections.

The first parameter is regarding how similar are the amounts of analogous parameters of different renewable resources; more specifically, the less different the ready times, deadlines, and unit penalty costs of different renewable resources, the more the degree of difficulty of the related instance. Roughly deducing, this is mainly because the difference between the objective function values of the various solutions is less remarkable for resources with more similarity, so it will be harder to distinguish the optimality of different alternatives in the middle stages of the solving process of the problem. Similar reasoning is considerable for the other parameter which regards the earliness value of the resources deadlines. Based on this parameter, the earlier the deadlines of the resources are, the more the computations required to solve the instances are.

Based on the aforementioned parameters affecting the degree of difficulty of the instances, we constructed four groups of instances. In order to do that, we generated a set of twenty instances as the base set and included it in every group. In addition, in each group we included several other sets whose instances were the same as the base set but differed in only one parameter affecting the degree of difficulty. These other instances are described in the following sections.

In order to have a full factorial design of the parameters that are related to the RCPSP, we chose sample problems of the base set from the well-known project scheduling library (PSPLIB) [43], but instead of using the RCPSP instances of this library, we used the multimode RCPSP (MRCPSP) instances, because they were generally subjected to less number of activities and renewable resources. We randomly selected twenty instances from the set j30 of the MRCPSP instances of the library and transformed each instance to an instance of the RCPSP-TWRTPC using the following method:

(i) In the instances of the set j30, each nondummy activity has three execution modes. To transform each selected instance, one mode was randomly chosen for each activity. Duration and the renewable resource requirement of the activity were set equal to the related amounts for the selected mode.

(ii) We randomly generated the ready time of the renewable resources using the normal distribution. We set the mean and standard deviation of the distribution equal to one-third and one-twelfth of the project critical path length, respectively.

(iii) We randomly generated the deadline of the renewable resources using the normal distribution. We set the mean and standard deviation of the distribution equal to three-fourths and one-fourth of the project critical path length, respectively.

(iv) We randomly generated the unit penalty cost of the renewable resources using the normal distribution. We set both of the mean and variance of the distribution equal to three.

5.2. Group I: Instances with Different Number of Activities. We included instances with different number of activities in the first group. We generated and included three sets in the same way of the generation of the base set, but instead of using the j30 set of the MRCPSP instances of the PSPLIB, we used instances with the same parameters but with 10, 20, and 60 activities. Relating to the instances with 10 and 20 activities, we used the j10 and j20 sets of the MRCPSP instances of the PSPLIB. Since no MRCPSP sample instance with more than 30 activities is available in the PSPLIB, we had to generate the related MRCPSP instances with 60 activities. We used the project generator software Progen to generate these instances.

Table 1 shows the number of instances solved to optimality within 10 seconds from each set in groups I, and Figure 1 shows the solving time of each instance by each solver. In

Table 1: Number of instances of each set of group I out of 20 solved to optimality in 10 seconds by the branch and bound algorithm and the CPLEX solver.

# activities in instance set	# instances solved by the CPLEX solver	# instances solved by the branch and bound algorithm
10	20	20
20	19	20
30	4	19
60	0	19

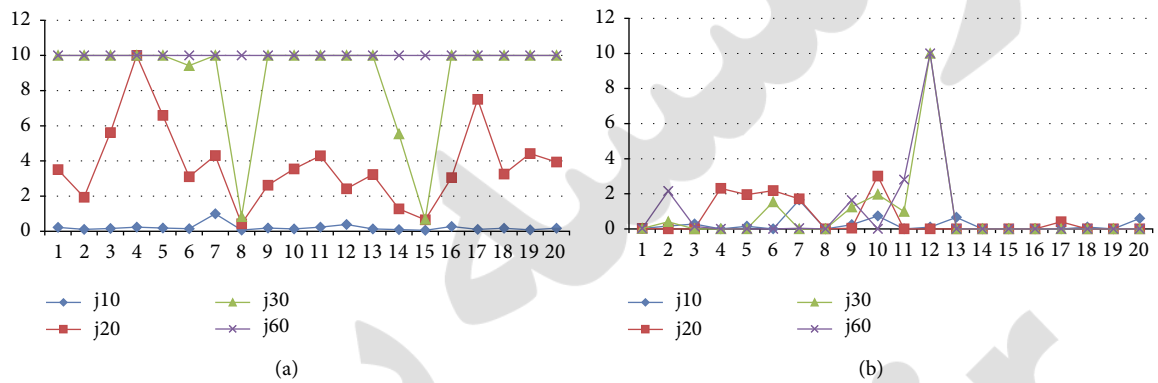

Figure 1: Solving time (seconds) of the instances of group I by (a) the CPLEX Solver and (b) the branch and bound algorithm.

this figure, the solving time of the unsolved instances in the time limitation has been shown to be 10 seconds. As the number of activities in the instances increases, according to the table, we can observe that the number of solved instances within 10 seconds decreases and also, according to the figure, the solving time of the instances generally increases. These trends imply a higher degree of difficulty for solving the problem as the number of activities increases, which is of course consistent with expectation. Comparing the results of the branch and bound algorithm with the results of the CPLEX solver in the table shows that they perform almost the same for the instances with 10 and 20 activities, but the branch and bound algorithm works remarkably better than the CPLEX solver for solving the instances with more number of activities. Also according to the figure, we can see that the branch and bound algorithm performs much faster than the CPLEX solver for most instances.

5.3. Group II: Instances with Different Number of Renewable Resources. We included instances with different number of renewable resources in the second group. We generated and included two sets other than the base set in this group. For the instances of the first set, one renewable resource was considered. We generated the instances of this set in the same way as the base set, but in each instance, we randomly removed one of the two renewable resource constraints. In the instances of the second included set other than the base set, four renewable resources were considered. Again we generated the instances of this set in the same way as the base set; however we extended the number of the renewable resource constraints. In order to do that, during the transformation of each selected MRCPSP instance to a RCPSP-TWRTPC instance of this set according to the process which was described in Section 5.1, instead of using the related resource requirement data of the activity under the selected mode, we used the data related to the unselected modes. The data related to each unselected mode were used as the resource requirements of the activity for two of the four resources. Also, the total availabilities of the third and fourth resources were set equal to the total availabilities of the first and second resources, respectively.

Table 2 shows the number of instances of each set of group II solved to optimality by the branch and bound algorithm and the CPLEX solver, and Figure 2 shows the solving time of each instance by each solver. In this figure, the solving time of the unsolved instances in the time limitation has been shown to be 10 seconds. As the number of renewable resources in the instances increases, we can observe that, according to the table, the number of problems solved to optimality by both solvers decreases and according to the figure, the solving time of each instance generally increases. These trends clearly show the direct impact of the number of the renewable resources in the computational requirements for solving the problem. Based on the results of the table, the branch and bound algorithm works remarkably better than the CPLEX solver for solving instances of the entire sets and the difference increases as the number of the renewable resources increases. Also, based on the figure, we can see that the branch and bound algorithm works much faster than the CPLEX solver for solving most instances.

5.4. Group III: Instances with Different Levels of Similarity of Renewable Resources Parameters Amounts. We included instances with different levels of similarity of the renewable resources parameters amount in the third group. In order to generate such different instances, we changed the variances of

Table 2: Number of instances of each set of group II out of 20 solved to optimality in 10 seconds by the branch and bound algorithm and the CPLEX solver.

# renewable resources in instance set	# instances solved by the CPLEX solver	# instances solved by the branch and bound algorithm
1	13	20
2	5	19
4	0	17

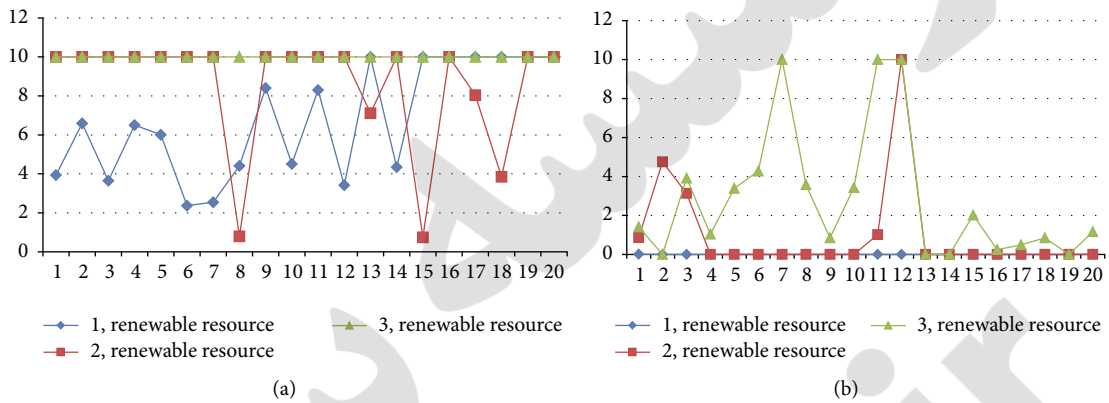

Figure 2: Solving time (seconds) of the instances of group II by (a) the CPLEX Solver and (b) the branch and bound algorithm.

the three distribution functions used in the generation of the ready time, deadline, and unit penalty cost of the renewable resources. The more the values of the related variances are, the less analogous the resources are expected to be. We generated and included two sets in the same way as the base set in group III, but in the first and second additional sets, we multiplied the base values of the three variances which were considered in the base set by 0.5 and 2, respectively.

Table 3 shows the number of instances of each set of group III solved to optimality by the branch and bound algorithm and the CPLEX solver, and Figure 3 shows the solving time of each instance by each solver. In this figure, the solving time of the unsolved instances in the time limitation has been shown to be 10 seconds. As instances with less analogous resources are targeted, we can observe that, based on the table, the number of problems solved to optimality by the CPLEX solver increases and based on the figure, the solving time of instances by both solvers generally decreases. These trends show the impact of the levels of similarity of the renewable resources parameters in the computational requirement for solving the problem. However, the table shows that the amount of changes in the levels considered in different instances here has no remarkable impact on the performance of the branch and bound algorithm.

Comparing the results of the branch and bound algorithm with the results of the CPLEX solver in Table 3 indicates that the branch and bound algorithm works remarkably better than the CPLEX solver for solving instances of the entire sets and the difference increases as the similarity of the renewable resources decreases. Also, based on Figure 3, we can see that the branch and bound algorithm works much faster than the CPLEX solver for solving most instances.

5.5. Group IV: Instances with Different Earliness Value of Renewable Resources Deadlines. We included instances with different earliness value of the renewable resources deadlines in the fourth group. In order to generate such different instances, we changed the mean of the distribution functions of deadline of the renewable resources. We generated and included two sets in the same way as the base set in the group IV, but in the first and second additional sets, we set the mean value of the resources deadlines equal to half and one time of the project critical path length, respectively.

Table 4 shows the number of instances of each set of group IV solved to optimality by the branch and bound algorithm and the CPLEX solver, and Figure 4 shows the solving time of each instance by each solver. In this figure, the solving time of the unsolved instances in the time limitation has been shown to be 10 seconds. As instances with earlier value of the renewable resources deadlines are targeted, we can observe that, based on the table, the number of problems solved to optimality by the CPLEX solver increases and based on the figure, the solving time of instances by both solvers generally decreases. These trends show the impact of this parameter in the computational requirement for solving the problem. However, the table shows that the amount of changes in this parameter has no remarkable impact on the performance of the branch and bound algorithm.

Comparing the results of the branch and bound algorithm with the results of the CPLEX solver in Table 4 indicates that

Table 3: Number of instances of each set of group III out of 20 solved to optimality in 10 seconds by the branch and bound algorithm and the CPLEX solver.

Instance set	# instances solved by the CPLEX solver	# instances solved by the branch and bound algorithm
First additional set	4	19
Base set	6	19
Second additional set	11	18

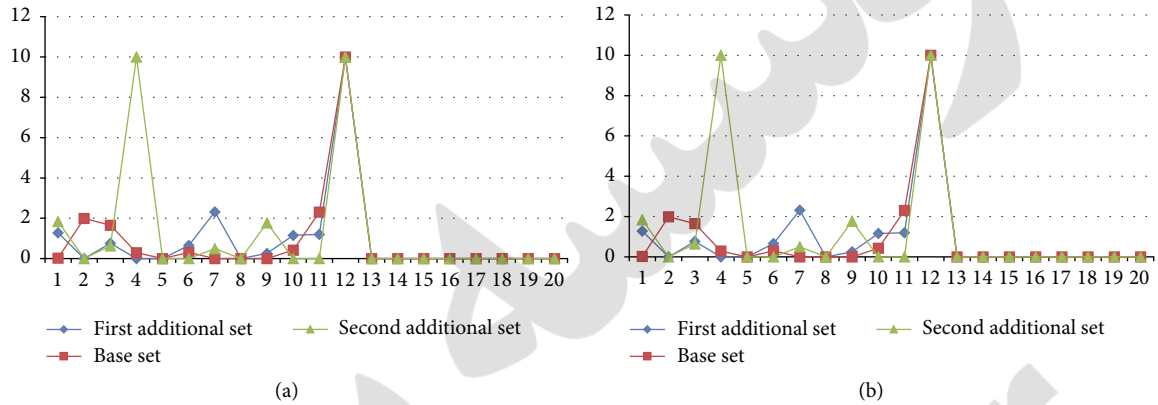

Figure 3: Solving time (seconds) of the instances of the group III by (a) the CPLEX Solver and (b) the branch and bound algorithm.

the branch and bound algorithm works remarkably better than the CPLEX solver for solving instances of the entire sets and the difference increases as the earliness value of the renewable resources deadlines decreases. Also, based on Figure 4, the branch and bound algorithm works much faster than the CPLEX solver for solving most instances.

6. Conclusions

In this paper we introduced and studied the RCPSP-TWRTPC problem. We customized a fundamental branch and bound method of RCPSP, precedence tree algorithm, for solving the problem. We also introduced several bounding and fathoming rules in the customized method, including the ones extracted from the original methods of RCPSP and used without any change or after customization and the ones designed specifically for the RCPSP-TWRTPC.

We performed a comprehensive computational experiment using the customized algorithm and reported the results. We also used the CPLEX solver in the analyses and comparisons. We generated and used different instances with different number of activities, number of renewable resources, level of similarity renewable resources parameters, and earliness value of renewable resources deadlines. Analyses showed that these three parameters affect the computational requirement for solving the problem and also the relative performance of the branch and bound algorithm and the CPLEX solver with respect to each other. The branch and bound algorithm revealed much better performance than the CPLEX solver by solving more instances of each set in the limited time and performing much faster for solving most instances, which implies its high efficiency in solving the RCPSP-TWRTPC problem.

More complicated branch and bound logics such as minimal delaying alternative can be studied in future works. Also, as the problem in question is NP-hard, devising nonexact or bounding methods would benefit handling instances, especially the large-sized ones.

Appendix

Assume that Sch_O is an optimum schedule for a given instance of the RCPSP-TWRTPC. Let AL_O be an activity list corresponding to the schedule Sch_O. In this activity list, activities are ordered in nondecreasing order of their start times in the schedule Sch_O and those activities with the same start times are ordered in increasing order of their index number. It is evident that since the schedule Sch_O is precedence feasible, the resulting activity list is precedence feasible too; that is, the order of each activity in the list is later than the order of every predecessor of the activity. So corresponding to this order, there must be a node like g in the last level of the branching tree of the instance, unless this node is not generated because of the dominance with another node. We prove that the schedule related to the node g, Sch_g, is an optimum solution for the problem. So whether node g is generated or dominated by another node, the optimum solution for the problem is gained in the branching tree. In order to prove this, we use deductive reasoning. We check the start time of the activities according to their order in AL_O and prove that the start time of each activity j in the schedule Sch_g, $S_{j,g}$, is not later than its start time in the schedule Sch_O, $S_{j,g'}$. So the related tardiness of each renewable resource in the schedule Sch_g cannot be more than the related value in the schedule Sch_O. First we check the start time of the first activity in AL_O. The first dummy activity of the project is

Table 4: Number of instances of each set of group IV out of 20 solved to optimality in 10 seconds by the branch and bound algorithm and the CPLEX solver.

Instance set	# instances solved by the CPLEX solver	# instances solved by the branch and bound algorithm
First additional set	3	19
Base set	7	19
Second additional set	16	18

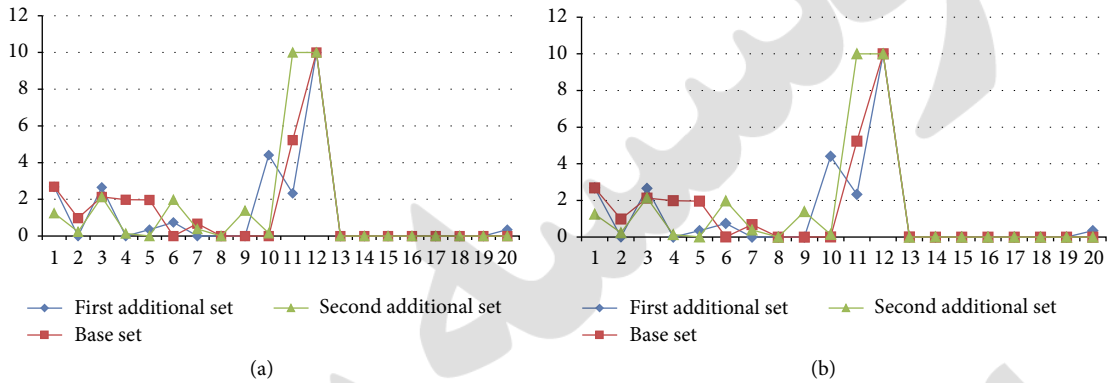

Figure 4: Solving time (seconds) of the instances of the group IV by (a) the CPLEX Solver and (b) the branch and bound algorithm.

the first activity in AL_O, because no activity can have earlier start time than this activity in any feasible period and it also has the least index number. According to the structure of the branch and bound algorithm, this activity is scheduled in the first period; that is, $S_{0,g} = 0$. So $S_{0,g'}$ cannot be less than $S_{0,g}$. Now we assume that, for every activity j in the position $1, \ldots, i-1$ of the activity list AL_O, $S_{j,g} \leq S_{j,g'}$ and we show that $S_{i,g} \leq S_{i,g'}$. According to the structure of the branch and bound algorithm, activity i is scheduled in the earliest feasible period. As $S_{i,g'}$ is a feasible period for scheduling i, it is scheduled either in this or in an earlier period, so $S_{i,g} \leq S_{i,g'}$.

Competing Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

References

[1] M. Ranjbar, M. Khalilzadeh, F. Kianfar, and K. Etminani, "An optimal procedure for minimizing total weighted resource tardiness penalty costs in the resource-constrained project scheduling problem," *Computers & Industrial Engineering*, vol. 62, no. 1, pp. 264–270, 2012.

[2] J. H. Patterson, R. Slowinski, F. B. Talbot, and J. Weglarz, "An algorithm for a general class of precedence and resource constrained scheduling problems," in *Advances in Project Scheduling*, R. Slowinski and J. Weglarz, Eds., pp. 3–28, Elsevier, Amsterdam, The Netherlands, 1989.

[3] J. A. Carruthers and A. Battersby, "Advances in critical path methods," *Journal of the Operational Research Society*, vol. 17, pp. 359–380, 1966.

[4] J. H. Patterson and G. W. Roth, "Scheduling a project under multiple resource constraints: a zero-one programming approach," *AIIE Trans*, vol. 8, no. 4, pp. 449–455, 1976.

[5] P. Brucker, S. Knust, A. Schoo, and O. Thiele, "A branch and bound algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 107, no. 2, pp. 272–288, 1998.

[6] H. R. Yoosefzadeh and H. R. Tareghian, "Hybrid solution method for resource-constrained project scheduling problem using a new schedule generator," *International Journal of Advanced Manufacturing Technology*, vol. 66, no. 5, pp. 1171–1180, 2013.

[7] U. Dorndorf, E. Pesch, and T. Phan-Huy, "A branch-and-bound algorithm for the resource-constrained project scheduling problem," *Mathematical Methods of Operations Research*, vol. 52, no. 3, pp. 413–439, 2000.

[8] J. Blazewicz, J. K. Lenstra, and A. H. G. R. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

[9] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch, "Resource-constrained project scheduling: notation, classification, models, and methods," *European Journal of Operational Research*, vol. 112, no. 1, pp. 3–41, 1999.

[10] K.-C. Ying, S.-W. Lin, and Z.-J. Lee, "Hybrid-directional planning: improving improvement heuristics for scheduling resource-constrained projects," *International Journal of Advanced Manufacturing Technology*, vol. 41, no. 3-4, pp. 358–366, 2009.

[11] R. Zamani, "A parallel complete anytime procedure for project scheduling under multiple resource constraints," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 1–4, pp. 353–362, 2010.

[12] R. Kolisch, "Efficient priority rules for the resource-constrained project scheduling problem," *Journal of Operations Management*, vol. 14, no. 3, pp. 179–192, 1996.

[13] M. Ranjbar, "Solving the resource-constrained project scheduling problem using filter-and-fan approach," *Applied Mathematics and Computation*, vol. 201, no. 1-2, pp. 313–318, 2008.

[14] V. Valls, F. Ballestín, and S. Quintanilla, "A hybrid genetic algorithm for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 185, no. 2, pp. 495–508, 2008.

[15] S. K. Shukla, Y. J. Son, and M. K. Tiwari, "Fuzzy-based adaptive sample-sort simulated annealing for resource-constrained project scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 36, no. 9-10, pp. 982–995, 2008.

[16] K. Bouleimen and H. Lecocq, "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version," *European Journal of Operational Research*, vol. 149, no. 2, pp. 268–281, 2003.

[17] P. R. Thomas and S. Salhi, "A tabu search approach for the resource constrained project scheduling problem," *Journal of Heuristics*, vol. 4, no. 2, pp. 123–139, 1998.

[18] K. Nonobe and T. Ibaraki, "Formulation and tabu search algorithm for the resource constrained project scheduling problem," in *Essays and Surveys in Meta-Heuristics*, C. C. Ribeiro and P. Hansen, Eds., pp. 557–588, Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.

[19] Q. Jia and Y. Seo, "An improved particle swarm optimization for the resource-constrained project scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 9, pp. 2627–2638, 2013.

[20] R. Agarwal, M. K. Tiwari, and S. K. Mukherjee, "Artificial immune system based approach for solving resource constraint project scheduling problem," *The International Journal of Advanced Manufacturing Technology*, vol. 34, no. 5-6, pp. 584–593, 2007.

[21] W. Herroelen, B. De Reyck, and E. Demeulemeester, "Resource-constrained project scheduling: a survey of recent developments," *Computers and Operations Research*, vol. 25, no. 4, pp. 279–302, 1998.

[22] S. Hartmann and R. Kolisch, "Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 127, no. 2, pp. 394–407, 2000.

[23] R. Kolisch and R. Padman, "An integrated survey of deterministic project scheduling," *Omega*, vol. 29, no. 3, pp. 249–272, 2001.

[24] R. Kolisch and S. Hartmann, "Experimental investigation of heuristics for resource-constrained project scheduling: an update," *European Journal of Operational Research*, vol. 174, no. 1, pp. 23–37, 2006.

[25] B. Abbasi, S. Shadrokh, and J. Arkat, "Bi-objective resource-constrained project scheduling with robustness and makespan criteria," *Applied Mathematics and Computation*, vol. 180, no. 1, pp. 146–152, 2006.

[26] B. Jarboui, N. Damak, P. Siarry, and A. Rebai, "A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems," *Applied Mathematics and Computation*, vol. 195, no. 1, pp. 299–308, 2008.

[27] H. Mokhtari, A. Aghaie, J. Rahimi, and A. Mozdgir, "Project time-cost trade-off scheduling: a hybrid optimization approach," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 5–8, pp. 811–822, 2010.

[28] A. Salmasnia, H. Mokhtari, and I. N. Kamal Abadi, "A robust scheduling of projects with time, cost, and quality considerations," *The International Journal of Advanced Manufacturing Technology*, vol. 60, no. 5-8, pp. 631–642, 2012.

[29] S. Li, Y. Jia, and J. Wang, "A discrete-event simulation approach with multiple-comparison procedure for stochastic resource-constrained project scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 63, no. 1–4, pp. 65–76, 2012.

[30] S. Kumanan, G. Jegan Jose, and K. Raja, "Multi-project scheduling using an heuristic and a genetic algorithm," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 3-4, pp. 360–366, 2006.

[31] M.-C. Wu and S.-H. Sun, "A project scheduling and staff assignment model considering learning effect," *The International Journal of Advanced Manufacturing Technology*, vol. 28, no. 11-12, pp. 1190–1195, 2006.

[32] H. Ke, W. Ma, and Y. Ni, "Optimization models and a GA-based algorithm for stochastic time-cost trade-off problem," *Applied Mathematics and Computation*, vol. 215, no. 1, pp. 308–313, 2009.

[33] H. Ke, W. Ma, and X. Chen, "Modeling stochastic project time–cost trade-offs with time-dependent activity durations," *Applied Mathematics and Computation*, vol. 218, no. 18, pp. 9462–9469, 2012.

[34] A. Azaron and R. Tavakkoli-Moghaddam, "A multi-objective resource allocation problem in dynamic PERT networks," *Applied Mathematics and Computation*, vol. 181, no. 1, pp. 163–174, 2006.

[35] A. A. Najafi and S. T. A. Niaki, "A genetic algorithm for resource investment problem with discounted cash flows," *Applied Mathematics and Computation*, vol. 183, no. 2, pp. 1057–1070, 2006.

[36] M. Ranjbar, F. Kianfar, and S. Shadrokh, "Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm," *Applied Mathematics and Computation*, vol. 196, no. 2, pp. 879–888, 2008.

[37] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, no. 1, pp. 1–14, 2010.

[38] J. Węglarz, J. Józefowska, M. Mika, and G. Waligóra, "Project scheduling with finite or infinite number of activity processing modes—a survey," *European Journal of Operational Research*, vol. 208, no. 3, pp. 177–205, 2011.

[39] M. Ranjbar, "A hybrid GRASP algorithm for minimizing total weighted resource tardiness penalty costs in scheduling of project networks," *International Journal of Industrial Engineering & Production Research*, vol. 23, no. 3, pp. 231–243, 2012.

[40] M. Khalilzadeh, F. Kianfar, A. Shirzadeh Chaleshtari, S. Shadrokh, and M. Ranjbar, "A modified PSO algorithm for minimizing the total costs of resources in MRCPSP," *Mathematical Problems in Engineering*, vol. 2012, Article ID 365697, 18 pages, 2012.

[41] J. Carlier and E. Pinson, "An algorithm for solving the jobshop problem," *Management Science*, vol. 35, pp. 164–176, 1989.

[42] E. Demeulemeester and W. Herroelen, *Project Scheduling a Research Handbook*, KLuwer Academic, New York, NY, USA, 2002.

[43] "Project Scheduling Library-PSPLIB," 2015, http://www.om-db.wi.tum.de/psplib/.