Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 634649, 15 pages
http://dx.doi.org/10.1155/2014/634649

Research Article

Branch and Bound Algorithms for Resource Constrained Project Scheduling Problem Subject to Nonrenewable Resources with Prescheduled Procurement

A. Shirzadeh Chaleshtarti,[1] S. Shadrokh,[1] and Y. Fathi[2]

[1] *Industrial Engineering Department, Sharif University of Technology, Azadi Street Tehran, Iran*
[2] *Edward P. Fitts Department of Industrial and System Engineering, North Carolina State University, Raleigh, NC, USA*

Correspondence should be addressed to A. Shirzadeh Chaleshtarti; a_shirzadeh@ie.sharif.edu

Received 14 November 2013; Accepted 6 January 2014; Published 27 February 2014

Academic Editor: Carla Roque

A lot of projects in real life are subject to some kinds of nonrenewable resources that are not exactly similar to the type defined in the project scheduling literature. The difference stems from the fact that, in those projects, contrary to the common assumption in the project scheduling literature, nonrenewable resources are not available in full amount at the beginning of the project, but they are procured along the project horizon. In this paper, we study this different type of nonrenewable resources. To that end, we extend the resource constrained project scheduling problem (RCPSP) by this resource type (RCPSP-NR) and customize four basic branch and bound algorithms of RCPSP for it, including precedence tree, extension alternatives, minimal delaying alternatives, and minimal forbidden sets. Several bounding and fathoming rules are introduced to the algorithms to shorten the enumeration process. We perform comprehensive experimental analysis using the four customized algorithms and also CPLEX solver.

1. Introduction

Resource constrained project scheduling problems (RCPSPs) are widely studied in the open literature. Several resource types have been defined in these problems. They are implied from various resources that are required by the practical projects. Renewable and nonrenewable resources are the most studied resource types in this field. Research charisma of these resources is consistent with their pervasive applications in practical cases. Nevertheless, a lot of the projects in real life are subject to some kinds of nonrenewable resources which are not exactly similar to the type defined in the literature. The reason is that, in the project scheduling literature, the availability of a nonrenewable resource is limited for the whole project life cycle; that is, the whole amount of the resource is available at the beginning of the project, but in practice this may not be the case. It is very common that some nonrenewable resources like project materials are procured along the project duration. These resources are not available in full amount at the beginning of the project and, neglecting the usage amounts from them, their availability increases along the time. Deteriorating nonrenewable resources and nonrenewable resources with a high amount of the holding or buying cost are usually subject to this case. For example, in large construction projects, high amount of beams may be required during the project. Procurement of the whole amount at the beginning of the project may not be possible due to the holding and handling issues of the beams. Also large amount of budget would be required in that case, which may not be available. Finally, this procurement policy may not be financially the best possible policy since it results in large amount of cost of capital. Considering these issue, a plan is usually considered for the procurement of the beams. Uncountable similar examples like this can be found in the real projects, which strongly motivate study of the subject.

In this paper, we study this different type of nonrenewable resources that are procured along the project duration according to some prescheduled plans. This type of resource is a special case of partially renewable resources [1]. Partially renewable resources are a generic resource type, which include renewable and nonrenewable resources as their special cases as well. The availabilities of partially renewable resources are associated with specific time intervals (subset of periods). On each interval related to each of these resources, total availability of the resource is specified. A project activity consumes a partially renewable resource only when the activity is processed partly or completely in one of the related intervals of the resource. The intervals of a partially renewable resource need to be neither disjoint nor conjunctive. Nonrenewable resources with prescheduled procurement can also be modeled by partially renewable resources. To this end, let us assume a nonrenewable resource l which has some prescheduled procurement plans. According to this plan, the availability of the resource from the beginning of the project planning horizon up to each period t is limited to an amount, say CR_{lt} ($CR_{lt} \leq CR_{l(t+1)}$). This condition can be modeled using a partially renewable resource Pl_t, whose related time interval includes periods of $[0, t]$ and its availability for this interval equals CR_{lt}. This same resource Pl_t can also be used to model the limited availability of the resource l from the beginning of the project up to any period $t' < t$, if $CR_{lt} = CR_{lt'}$. So supposing that T is the number of periods of project planning horizon, the resource l can be completely substituted using a set of partially renewable resources Pl. Each of the resources of this set is characterized as shown previously to indicate the limited availability of the resource l from the first period of the planning horizon up to a period $t \in \{\tau \mid 0 \leq \tau \leq T, CR_{l\tau} < CR_{l(\tau+1)}\}$.

We extend the standard form of the RCPSP problem by subjecting the problem to nonrenewable resources with prescheduled procurement. We call the extended problem resource constrained project scheduling problem, extended with prescheduled nonrenewable resources (RCPSP-NR). In order to solve this problem, we customize four basic branch and bound approaches of RCPSP for RCPSP-NR. These algorithms include precedence tree, extension alternatives, minimal delaying alternatives, and minimal forbidden sets. Several bounding, fathoming, and dominance rules are introduced to these algorithms to shorten the enumeration process. The developed algorithms are used to perform comprehensive experimental analysis. The CPLEX solver is also used in the experiments and comparisons.

The remaining sections of this paper are organized as follows. In the next section, we review some of the previous work related to this paper. In Section 3, RCPSP-NR is described in detail. In Section 4, basic scheme of the branch and bound algorithms of this paper is specified. In Sections 5, 6, 7, and 8, methods of precedence tree, extension alternatives, minimal delaying alternatives, and minimal forbidden sets are described, respectively. In each of these four sections, we first review the original method for RCPSP, followed by the description of the appropriate modifications made in the method to work for RCPSP-NR. Section 9 is dedicated to computational analysis on all algorithms and finally Section 10 concludes the whole work.

2. Related Work

The RCPSP problem in its standard form is subject to the renewable resources. The literature on this problem dates back to the 1960s [2]. Many solving and bounding algorithms have been developed for the problem. It has also been extended in different ways, such as the extension to minimum and maximum time lags [3], multiple objectives [4], and multiple decentralized projects [5]. The literature on the problem and its extensions have been well reviewed in many papers, such as [6–10].

Nonrenewable resources, on the other hand, are basically studied in the multimode project scheduling problems. Time-cost tradeoff problem (TCTP) and multimode RCPSP (MRCPSP) are the most studied problems in this area. Albeit, not all the studies on the MRCPSP consider nonrenewable resources in addition to the renewable resources. Some of the recent studies on the problem subject to both resource constraints are discussed in [11–15].

In the TCTP problem, each duration time of every activity is related to some cost that is resulted from the nonrenewable resource usage of the activity. The trade-off between duration time and cost may be discrete [16, 17] or continuous [18]. Akkan et al. [19] extensively review the literature on the discrete version of the problem (DTCTP). The continuous version of the TCTP (CTCTP) has not been studied as much as the DTCTP. We refer the readers to [20] for a comprehensive survey in this regard.

To the extent of our knowledge, nonrenewable resources with prescheduled procurement have not been specifically studied in the project scheduling literature. Yet the generic form of these resources, partially renewable resources, has been studied in the literature. Böttcher et al. [1] introduced partially renewable resources and described some instances of their applications in timetabling and shift scheduling aspects. They generalized RCPSP by using partially renewable resources in the problem and called the extended problem RCPSP/π. References [21, 22] extended this work by presenting more capabilities and applications of partially renewable resources. Reference [21] also proposed several groups of inexact algorithms for solving RCPSP/π. For this same problem, metaheuristic algorithms are proposed in [23, 24]. Zhu et al. [25] integrated partially renewable resources in the MRCPSP and proposed a branch and cut algorithm for solving this problem.

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 0 and $n + 1$ as initial and terminal nodes, respectively. The processing time and resource usage of these dummy activities equal zero. Let $K = \{1, \dots, NB\}$ and $L = \{1, \dots, NC\}$ be the sets of renewable and nonrenewable resources, respectively. Each activity j ($j = 0, \dots, n + 1$) 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 and requires c_{jl} units of nonrenewable resource l ($l \in L$) which are used in the first period of its execution. Besides, activity j has a set of immediate predecessor activities, P_j. Every renewable

```
(1) Perform preprocessing and stop if the instance is infeasible
(2) Perform algorithm preprocesses (specific to minimal forbidden sets algorithm)
(3) Determine disjunctive pairs of activities
(4) Specify the initial upper bound
(5) Determine resources periodic availability profiles
(6) Generate the initial node and select it for branching
(7) Branch selected node (specific structure for each algorithm)
(8) Fathom each new node containing feasible solution and update upper bound and LST of activities if necessary
(9) If there is at least one node not branched or fathomed yet, select a new node for branching; Otherwise report the best
    feasible solution achieved and stop
(10) Perform fathoming check(s) on the selected node for branching and continue from Step 7 if the node is not fathomed;
    otherwise continue from Step 9
```

Algorithm 1: Basic scheme of the branch and bound algorithms.

resource k has constant availability of R_k for each period over the project duration. Nonrenewable resources are not ready in full amount at the beginning of the project, but they are procured along the project according to prespecified plan, so that every nonrenewable resource l has availability of CR_{lt} from the beginning of the project up to the period t ($CR_{lt} \geq CR_{l(t-1)}$). No preemption is permitted during the execution of activities; all activities have single mode, and they are ready at the beginning of the project horizon. All parameters are deterministic and integers. The problem is to find the start time of each activity j, S_j, ($j = 0, \ldots, n+1$), such that all problem constraints are satisfied and the project makespan is minimized.

Suppose that the earliest and latest start times of each activity j, EST_j and LST_j, ($j = 0, \ldots, n+1$), are determined with forward and backward passes. In order to do that, we start by fixing the value of LST_{n+1} and the latest finish time of the last dummy activity (LFT_{n+1}) is equal to T, where T is an upper bound for optimum project makespan. T can be determined by any valid method. Two methods in this regard are later proposed in Section 4.4 of this paper. We now define the following decision variable:

$$x_{jt} = \begin{cases} 1 & \text{if activity } j \text{ is started in period } t \\ 0 & \text{otherwise,} \end{cases}, \quad j = 1, \ldots, n+1, \ t = \text{EST}_j, \ldots, \text{LST}_j. \tag{1}$$

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

$$\text{Min} \quad \sum_{t=\text{EST}_{n+1}}^{\text{LST}_{n+1}} t x_{(n+1)t}, \tag{2}$$

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

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

$$\sum_{j=1}^{n} \sum_{\tau=t-d_j+1}^{\min(t,\text{LST}_j)} r_{jk} x_{j\tau} \leq R_k, \quad k \in K, \ t = 1, \ldots, \text{LST}_n, \tag{5}$$

$$\sum_{j=1}^{n} \sum_{\tau=\text{EST}_j}^{\min(t,\text{LST}_j)} c_{jl} x_{j\tau} \leq CR_{lt}, \quad l \in L, \ t = 1, \ldots, \text{LST}_n, \tag{6}$$

$$x_{jt} \in \{0, 1\}, \quad j = 1, \ldots, n+1, \quad t = \text{EST}_j, \ldots, \text{LST}_j. \tag{7}$$

Objective (2) is to minimize the project makespan. Constraint (3) guarantees that each activity j can only have a single start time from the period $[\text{EST}_j, \text{LST}_j]$. Constraint (4) takes into consideration precedence relations between each pair of activities (i, j), where i is an immediate predecessor of j. Constrains (5) and (6) regard renewable and nonrenewable resources usage limitations, respectively and, finally, constraint (7) denotes the domain of variables.

By omitting constraint set 5 from the model above, it turns to a formulation for the RCPSP. That formulation is similar to the one from Pritsker et al. [26] for the RCPSP, except that, in the Pritsker et al. [26] formulation, decision variables, say x'_{jt} ($j = 1, \ldots, n+1$, $t = \text{EST}_j, \ldots, \text{LST}_j$), specify finish time of activities. Both formulations would be the same if the variables of either of them are changed according to the relation $\forall j, t : x'_{jt} = x_{jt} + d_j - 1$.

4. Basic Scheme of the Algorithms

For the collection of four branch and bound algorithms that we develop here, we use the same basic scheme, shown in Algorithm 1. In this section we describe this scheme.

```
(i)   Determine SIUB
(ii)  $LST_{n+1} = LFT_{n+1} = SIUB$
(iii) Determine LST and LFT of the activities using backward pass
(iv)  Compute LSTLFT of each activity and generate AL by prioritizing the activities in non-decreasing order of their LSTLFTs;
      If tie happens, prioritize the activities related to the tie in increasing order of their index numbers
(v)   Schedule activities in their related order of the AL in the earliest precedence and resource feasible times
(vi)  Report the finish time of last project activity as modified initial upper bound
```

Algorithm 2: Pseudocode of modified initial upper bound determination for RCPSP-NR.

4.1. Preprocessing. There are two cases in which a given instance has no feasible solution. The first case is where we have shortage of renewable resources. In this case there exist 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. The second case is where we have a shortage of nonrenewable resources. In this case there exists a nonrenewable resource l in the problem whose availability up to the last period of its procurement plan is less than the total required amount by the entire project activities. If none of these two states exists in a given instance, there are feasible solutions and the instance is feasible. So the branch and bound algorithms check the feasibility of the instance in the first step.

4.2. Algorithm Preprocesses. This step is specific to the minimal forbidden sets algorithm which is described in detail later in Section 8.

4.3. Disjunctive 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 determine some execution priorities between them.

Interval-Based Disjunctive Consistency Test [27]. For two disjunctive activities of i and j, if $\text{LST}_i\text{-EST}_j < d_j$, i must precede j.

We perform this test during the fathoming check at every node of the branch and bound 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 their activities.

4.4. Initial Upper Bound Specification. A rather simple procedure to specify an initial upper bound for RCPSP-NR is that the duration of all project activities is summed and added to the earliest period in which all nonrenewable resources necessary for project execution are ready. We call this upper bound, simple initial upper bound (SIUB) and improve it to get a tighter one which we call modified initial upper bound (MIUB). In order to determine MIUB, we generate an activity list (AL) and schedule activities based on the list. The related makespan of the generated schedule specifies MIUB. Activities of the list are selected in their related order and each selected activity is scheduled in the earliest feasible time considering the precedence and resource constraints. In order to generate a rather good activity list, we prioritize them in a nondecreasing order of the sum of their latest start and finish times (LSTLFT). In case a tie happens where the LSTLFT values for two or more activities are equal, we prioritize them in increasing order of their index numbers. The reason for using LSTLFT quantity in here is explained as follows. In [13], several of the most efficient heuristics for minimization of project makespan were compared. Albeit the problem under study was the multimode RCPSP (MRCPSP), but, during the solution methods, activities mode assignment were first accomplished, turning the problem to RCPSP, and then scheduling heuristics were applied. According to comparisons, LSTLFT rule concluded the best among other single-pass heuristics for prioritizing activities. This method has also been compared to the best multipass methods. Results showed that multipass methods needed much more time than single-pass ones; however they usually resulted in negligible improvement in the solution. Therefore, considering time requirements, LSTLFT choice seems the most efficient one among those compared.

In order to determine LST and LFT of activities, we fix the LST_{n+1} and LFT_{n+1} equal to SIUB and then we perform a backward pass. So, the MIUB determination procedure can be summarized as the pseudocode shown in Algorithm 2.

4.5. Resources Availability Profiles. For each renewable and nonrenewable resource in the problem, we generate a resource availability profile which shows the periodic availability of the resource from the first up to the upper bound period. These profiles are used in the branch and bound algorithms and updated by scheduling each activity. Note that the requirement of each nonrenewable resource l for project execution is procured up to period $\eta_l = \min\{t : CR_{lt} \geq \sum_{j=1}^{n} c_{jl}\}$. So we keep track of availability of each nonrenewable resource l up to period η_l.

4.6. Initial Node Generation and Branching. The branching tree of the algorithms is initiated with the initial node. This node is generated and branched as the only available node according to specific structure of each algorithm.

4.7. Branching. Each selected node for branching is branched according to the structure of the specific branch and bound method.

```
(1) Fix EST of scheduled activities
(2) Determine PBEST of unscheduled activities
(3) Determine EST of unscheduled activities equal to their RBEST
(4) Terminate lower bound determination process and fathom the node if for an activity LST < EST
(5) Perform interval-based disjunctive consistency test and continue the process from Step 2 if at least one new precedence
    relation is introduced by the test
(6) Perform unit interval consistency test and if a resource shortage happens, terminate lower bound determination process and
    fathom the node
(7) Report start time of the last dummy activity as critical path based lower bound
```

Algorithm 3: Pseudocode of lower bound determination method based on modified critical path length.

4.8. Fathoming Feasible Nodes and Upper Bound Update. After each branching, each newly generated node like g that contains a feasible solution is fathomed and, if the makespan of its related schedule, say T_g, is less than the current upper bound, upper bound is updated as T_g. In the rest of algorithm, the related makespan of a better solution has to be at most $T_g - 1$. In order to meet this threshold, LST and LFT of each activity can be determined using backward pass. These quantities are used in fathoming check of the nodes.

4.9. Node Selection for Branching. In all algorithms, we use depth-first-search method to keep the memory requirements low. According to this method, an unfathomed 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 number of scheduled activities, a node with the least associated lower bound, and the node with the least index number.

4.10. Fathoming Check. Fathoming check is performed on each unfathomed node g selected for branching. In order to do that, a lower bound on the related project makespan of g, LB_g, is determined. According to the structure of the algorithms, each new offspring node generated by branching node g will not have a lower makespan than LB_g. So if LB_g exceeds the current upper bound, no better solution can be obtained by branching node g and the node is fathomed.

In addition to fathoming based on lower bound, in each algorithm depending on its structure, some dominance rules are used to possibly fathom the nodes whose corresponding schedule are not promising. For each dominance rule, it is proved that the nodes fathomed using the rule cannot result in better solutions than some other nodes available in the branching tree. As these rules depend on the algorithm, we describe each in the context of the related algorithm.

The effectiveness of both fathoming tests as well as the dominance tests for many of the rules depends on the current value of the upper bound. Tighter upper bound values lead to more effectiveness in both tests. Since the upper bound may be improved during the algorithms, 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.

In the following we describe how the makespan lower bound for each node is determined.

4.10.1. Makespan Lower Bound Determination. A simple lower bound on project makespan can be determined using renewable resource constraints. This method is based on the fact that total availability of each renewable resource during the whole project execution time cannot be less than resource amount required by all activities. So this lower bound is determined as [20]

$$\max_{k\in K}\left\{\sum_{j=1}^{n}\frac{r_{jk}\cdot d_j}{R_k}\right\}. \tag{8}$$

We determine another lower bound for each node equal to the project critical path length; however we make the following modifications to this method to get better bounds. So the related lower bound of each node will be the maximum of lower bounds gained based on renewable resources and the critical path length.

Algorithm 3 shows the pseudocode of lower bound determination method based on modified critical path length. Some details of this method are described in the following.

(i) *EST Determination.* We fix the earliest start time (EST) of scheduled activities equal to their start times. For the rest of the activities, EST can be determined using forward pass. However another lower bound on the start time of each activity is achievable based on the resource constraints. To distinguish these two lower bounds, we call the EST of each activity gained from forward pass method and resource constraints precedence based earliest start time (PBEST) and resource based earliest start time (RBEST), respectively. RBEST for an activity is the earliest time equal or later than PBEST in which necessary amounts of all resources for execution of activity are available. RBEST of each activity is determined using the current resource availability profiles.

(ii) *Unit Interval Consistency Test* [28]. This test is based on the fact that each activity j will be definitely in progress during the periods of $[\mathrm{LST}_j, \mathrm{EFT}_j = (\mathrm{EST}_j + d_j - 1)]$. According to this, we decrease both renewable and nonrenewable resource requirements of activity

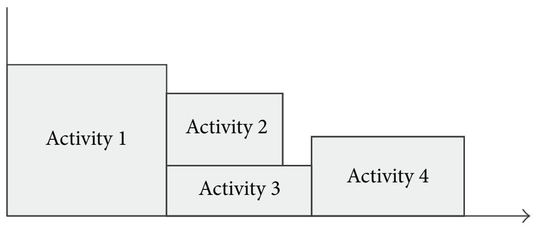


Figure 1: Example of a partial schedule with two different activity lists.

from the resource availability profiles. If, in some period, the availability of some resource gets negative, the node cannot result in a feasible schedule with makespan better than the current upper bound. So the node is fathomed.

5. Precedence Tree Algorithm (PTA)

In this section, we briefly describe the precedence tree algorithm for RCPSP. Then we describe the modifications we make to the algorithm to work for RCPSP-NR.

5.1. Basic Algorithm for RCPSP. Most of branch and bound algorithms for RCPSP use partial schedules which are associated with nodes in the branching tree. The difference among them regards to the branching and pruning methods. One of the methods using this approach is the one proposed by [29]. In this algorithm, partial schedules containing parts of the project activities are completed along the branching tree. Regarding to each selected node g for branching, the set of eligible activities (EJ_g) for this node 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 precedence and resource constraints.

Several dominance rules may be used in this algorithm. Here we introduce two rules from [20] that can be used in RCPSP-NR as well. Regarding to each partial schedule in each node, we can specify an activity list that contains activities in the order of their start times. Sometimes it is possible to relate more than one activity list to a given partial schedule. For example, for the partial schedule shown in Figure 1, we can have either activity list (1, 2, 3, and 4) or (1, 3, 2, and 4). 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 similar solutions. Note that, 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 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 $j < i$.

The two dominance rules mentioned above can be used to tighten critical path based lower bound as well. According to them, if i is the last scheduled activity, for each unscheduled activity j, $S_i \leq S_j$ if $i < j$ and $S_i + 1 \leq S_j$ if $i > j$; otherwise the related node is dominated and fathomed.

5.2. Customized Algorithm for RCPSP-NR. To customize the precedence tree algorithm introduced in Section 5.1 for RCPSP-NR, we make the following modifications to it and use it in the context described in Section 4.

(i) In each newly generated node, the new activity is scheduled in the first feasible period that other than precedence and renewable resource feasibility has nonrenewable resource feasibility; that is, sufficient nonrenewable resources exist for activity execution.

(ii) Based on the two dominance rules, in each node like g, if i is the last scheduled activity, for each unscheduled activity like j, $S_i \leq S_j$. On the other hand, a lower bound on the minimum required time for execution of unscheduled activities can be determined based on their renewable resource usage, which is equal to

$$\max_{k \in K} \left\{ \sum_{j \in \text{set of unscheduled activities}} \frac{r_{jk} \cdot d_j}{R_k} \right\}. \tag{9}$$

So a lower bound on the makespan can be determined by adding $S_i - 1$ to the amount gained in (9). We use this method in addition to the one described in Section 4.10.1, so the lower bound for each node will be the maximum amount gained by these two methods.

6. Extension Alternatives Algorithm (EAA)

In this section, we briefly describe the extension alternatives algorithm for RCPSP. Then we describe the modifications we make to the algorithm to work for RCPSP-NR.

6.1. Basic Algorithm for RCPSP. This method was proposed by Stinson et al. [30] for RCPSP. It is based on partial schedules similar to the PTA, but, instead of scheduling just one activity in each time, a group of activities are selected to be scheduled. In this approach, each node g of the branching tree is associated with a decision point t_g and, considering precedence and resource constraints, a subset of eligible activities (EJ_g) called extension alternative (EA_g) is to be selected for execution in t_g. Extension alternatives must contain all activities in process in the decision point and must be nonempty in order to secure algorithm termination. t_g has to be later than the decision point of the parent node, t_{g-1}, and it is determined as the earliest finish time of currently scheduled activities, because in these points some renewable resources turn vacant and also successors of already finished activities may be able to be scheduled without violating precedence constraints.

The following two dominance rules are applicable in this algorithm.

Dominance Rule 1. Assuming an extension alternative EA_g and an activity i, $i \in EJ_g$, $i \notin EA_g$, and $d_i \leq d_j$ for each activity $j \in EJ_g$. If i can be included in EA_g without violating resource constraints, EA_g is dominated with another extension alternative with the same activities as EA_g plus i. In other words, it is sufficient to only check maximal extension alternatives.

Dominance Rule 2. Any node g associated with the decision point t_g is dominated if there exists an activity i ($i \in EJ_g$) that could be included in the extension alternative of the parent node; that is, i can start earlier than t_g. This dominance rule is called left shift dominance rule [31].

Critical path based lower bound can be used for fathoming in this algorithm.

6.2. Customized Algorithm for RCPSP-NR. We customize this method for RCPSP-NR with the following modifications.

(i) Besides precedence and renewable resource constraints, nonrenewable resources are to be considered to develop extension alternatives; that is, in each decision point t_g, only those activities can be selected for an extension alternative and scheduled that, in addition to precedence and renewable resource feasibility, nonrenewable resource constraints remain feasible.

(ii) In each time point that availability of some nonrenewable resource increases, new activities may be able to be scheduled. So in order to determine the decision point of each node, these points are to be considered in addition to the finish times of scheduled activities; that is, the decision point is determined as the earliest point later than the decision point of the parent node that either at least one of the currently scheduled activities finishes or availability of at least one nonrenewable resource increases.

(iii) Contrary to RCPSP, it is feasible to have extension alternatives without any activity for RCPSP-NR. Two reasons can be given regarding to this. Firstly, in a decision point, no activity may be scheduled because of the lack of nonrenewable resources. Secondly, in RCPSP regarding to some activity $i \in EJ_g$, no new opportunity appears for scheduling other activities by delaying i, but in RCPSP-NR, nonrenewable resources necessary for i remain unused for the latter periods and, augmenting with procured amounts in those later periods, some activities may be able to be scheduled relatively sooner. However these cases will not be valid if there exists some activity $i \in EJ_g$ that uses either no nonrenewable resource or some nonrenewable resources like l for which $CR_{lT} - CR_{lt_g} = 0$.

(iv) In each node g, a lower bound on the EST of each unscheduled activity is determined equal to t_g. Using this and renewable resource usage of unscheduled activities, a lower bound on the related makespan of the node can also be determined. According to renewable resource usage, the minimum required time for scheduling of unscheduled activities is determined by (9) and, adding this amount to t_g, the lower bound is specified.

7. Minimal Delaying Alternatives Algorithm (MDAA)

In this section, we briefly describe the minimal delaying alternative algorithm for RCPSP. Then we describe the modifications we make to the algorithm to work for RCPSP-NR.

7.1. Basic Algorithm for RCPSP. This algorithm is based on the concept of delaying alternatives used by Christofides et al. [32] which was enhanced by Demeulemeester and Herroelen [33]. In this algorithm, activities are scheduled under the precedence constraints until a resource conflict happens; that is, the amount of some required resources needed for executing activities in a period exceed the availability of the resource. In this case, the conflict is to be resolved, naturally by delaying some sets of activities in progress in the period, called delaying alternative. This idea of the algorithm is opposite to PTA and EAA, such that all possibilities of scheduling activities are considered just in case a resource conflict might show up later.

Demeulemeester and Herroelen [33] showed that only minimal delaying alternatives are needed to be considered to resolve a resource conflict; that is, all delaying alternatives for each of which no proper subset are a valid delaying alternative. This is mainly because the other activities can be delayed at subsequent levels of the branching tree. Activities of a selected minimal delaying alternative are delayed to the earliest finish time of an undelayed activity in progress. Left shift dominance rule is applicable in this algorithm and the critical path based lower bound method is used for fathoming.

7.2. Customized Algorithm for RCPSP-NR. We customize this method for RCPSP-NR with the following modifications.

(i) In addition to renewable resource conflicts, nonrenewable resource shortages are to be considered here and resolved by generating and delaying the related minimal delaying alternatives. For this case, in any period like t that availability of a nonrenewable resource l becomes a negative amount like A_{lt}, we define D_t as the set of activities started by or at t according to the schedule; then each delaying alternative DA_t is a subset of D_t in the way that $\sum_{j\in DA_t} c_{jl} \geq |A_{lt}|$. Also, each minimal delaying alternative MDA_t is a delaying alternative that none of its proper subsets is a delaying alternative. Note that, if a predecessor of an activity i ($i \in D_t, i \notin \mathrm{MDA}_t$) like

j is embedded in MDA_t, delaying activities of MDA_t cause delaying i as well, so i can be regarded as a member of MDA_t implicitly and, based on this, MDA_t may not be a minimal delaying alternative but only a delaying alternative. So we force another limitation on determination of minimal delaying alternatives here that an activity i $(i \in D_t)$ can be included in a minimal delaying alternative MDA_t only if all of its successor that are present in D_t are included in MDA_t too.

(ii) Activities of an MDA_t are delayed to a period $t' > t$ for which $CR_{lt'} - CR_{lt} \geq A_{lt}$. Delaying is made by forcing constraints $t' \leq \mathrm{PBEST}_j$ for each $j \in \mathrm{MDA}_t$.

(iii) For an $i \in \mathrm{MDA}_t$, if $t' > \mathrm{LST}_i$, the generated node will not result in a better solution than current upper bound. So such test is performed for each MDA and extra ones are omitted. If no MDA remains, the current node is fathomed.

(iv) In each node g, as start time of all activities except from the first dummy one can change by further branching until g is fathomed, all activities are assumed as unscheduled ones and lower bound determination method and fathoming check are performed on this basis. However note that the process remains effective through algorithm, since new lower bound constraints are made on PBEST of activities and LSTs are updated as well.

(v) A renewable resource based lower bound on requirement is also determined equal to

$$t - 1 + \max_{k \in K} \left\{ \sum_{\forall j : \mathrm{EST}_j \geq t} \frac{r_{jk} \cdot d_j}{R_k} \right\}. \tag{10}$$

(vi) Note that in both cases of renewable and nonrenewable resource shortages, as activities are delayed from a period t to a latter period, no new conflict occurs by period t. So, to keep the required memory low and fasten the algorithm, for each new node generated to resolve a conflict at t, we only keep track of periodic resource availabilities in period t and later ones up to T.

8. Minimal Forbidden Sets Algorithm (MFSA)

In this section, we briefly describe the minimal forbidden sets algorithm for RCPSP. Then we describe the modifications we make to the algorithm to work for RCPSP-NR.

8.1. Basic Algorithm for RCPSP. Igelmund and Radermacher [34] introduced a branch and bound algorithm based on minimal forbidden sets for RCPSP. A Forbidden set is a set of activities for which no precedence relations exist between any two of them and their total resource requirement for at least one renewable resource is more than the availability of that resource. A minimal forbidden set (MFS) is also defined as a forbidden set that has no other forbidden sets as its proper subset. So no activity can be omitted from an MFS so that it still remains as an MFS. In order to have a feasible schedule in RCPSP problem, all activities of one MFS cannot be concurrently executed. In order to force this condition, at least one precedence relation is to be introduced between two activities of an MFS so that the MFS is broken. Therefore the base of this branch and bound algorithm is the introduction of precedence relations between activities of each MFS so that all these sets are broken. This idea of the algorithm is similar to the MDAA which tries to resolve resource usage conflicts.

In this algorithm, in each newly generated node, a new precedence relation is introduced between two activities so that at least one MFS is broken. Activities are scheduled in nodes considering their precedence relations using forward pass. If the related node is still not feasible regarding to renewable resources, critical path length specifies a lower bound for the project makespan and is used for fathoming check. Having selected a node for branching, an MFS not broken according to the selected node is to be selected to be broken. In order to do that, a value is specified for each new precedence relation between each two project activities. This value is equal to the number of MFSs that will be broken in the node by adding that relation. But it is noticed that, between some activities, precedence relations already exist and no more relation can be introduced, so such infeasible relations are not considered. Also new precedence relations that cause generating schedules with makespan more than the current upper bound are not considered. The other point is that the introduction of some precedence relations can implicitly break an MFS; that is, given two activities of i and j in an MFS, if according to a relation, i or a successor of that is set as the predecessor of j or one of its predecessors and i would be the predecessor of activity j (immediately or nonimmediately), so the related MFS is broken.

Having valued new feasible precedence relations between activities, a weight is assigned to each MFS. This weight is equal to the sum of values of introducing new relations between each two activities of that MFS. According to these weights, one MFS with the highest weight is selected from the ones not broken yet in the node and branching is performed by introducing feasible precedence relations between each two activities of the selected MFS.

The only remaining point is that in some nodes although no precedence relations have been introduced for breaking some MFSs, neither in the node nor in its parents, the related schedule of the node is in a way that activities of these MFSs have no overlap during their executions; therefore these MFSs are implicitly broken, the schedule is feasible, and the node is fathomed.

8.2. Customized Algorithm for RCPSP-NR. Our customized minimal forbidden sets algorithm for RCPSP-NR is a hybrid algorithm of MFSA and MDAA. In this algorithm, we initially neglect renewable resource conflicts and use a similar method as the original one for RCPSP to resolve all conflicts in these resources. In this part, in order to determine the lower bound of makespan, all activities are considered unscheduled, because their start time can change in progress. For each node, in which no renewable resource conflict exists anymore, we focus on resolving possible shortcomings of nonrenewable resources. We check the inventory of all these resources in order of time and regarding to each period in which a shortcoming exists; we use the same procedure as the one described in Section 7.2 to resolve the shortcoming. The noticeable point here is that, on performing this delaying action, MFSs which have been broken in the node still remain broken, but the ones that have been implicitly broken according to the schedule may no longer be implicitly broken. So for any node generated via branching of the parent node using MDAA branching rule, we check the feasibility of the node based on the renewable resources at first and resolve possible conflicts in this part via branching based on minimal forbidden sets and then we check for possible nonrenewable resources shortcomings.

9. Experimental Analysis

In this part we present the results of a comprehensive computational experiment that we conducted regarding the algorithms presented in this paper for RCPSP-NR. All algorithms were 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 we first describe the sample problems that we used and then we present and discuss the results categorized in four parts.

We used all four algorithms and also the CPLEX solver to solve each sample problem. We imposed a time limit of 20 seconds on the execution time for solving each instance by each algorithm and by the CPLEX solver, so that we could complete the experiments within a reasonable time. This time limit has been selected empirically by try and error to well illustrate the performance differences among the algorithms and also the CPLEX solver. In order to evaluate the effectiveness of the algorithms for solving RCPSP-NR, we used two metrics. The first metric is the number of instances solved within 20 seconds. The analyses of Sections 9.2, 9.3, and 9.4 are based on this metric. The second metric is the actual execution time for solving every instance that was solved within 20 seconds. The analyses of Section 9.5 are based on this metric.

9.1. Sample Problems. As RCPSP-NR is an extension of RCPSP, all parameters that affect the degree of difficulty of an instance of 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 RCPSP-NR too. Besides, the additional parameters regarding the nonrenewable resources may also affect the degree of difficulty of the RCPSP-NR instances. Here one obvious parameter is the number of nonrenewable resources in the problem. The other parameter is η_l as defined in Section 4.5 for each nonrenewable resource l. The larger the η_l is, the more periods along project planning horizon are to be checked for the nonrenewable resource l constraint, therefore the more is the computational effort needed. Based on these two parameters, we constructed two groups of sample problems. In the first

Table 1: Value of the parameters c_{jl} in test instances.

Number of resources	Parameter	Selected value for parameter
1	c_{j_1}	$n_{jm_1 1}$
2	c_{j_1}	$n_{jm_1 1}$
	c_{j_2}	$n_{jm_1 2}$
4	c_{j_1}	$n_{jm_2 1}$
	c_{j_2}	$n_{jm_2 2}$
	c_{j_3}	$n_{jm_3 1}$
	c_{j_4}	$n_{jm_3 2}$

group we fixed the number of nonrenewable resources and changed the average value of η and in the second group we did the reverse; that is, we fixed the average value of η and changed the number of nonrenewable resources. The impacts of these parameters on the computational requirements of various algorithms and also on the completion time of the related projects are experimentally viewed in Sections 9.3 and 9.4.

In order to have a full factorial design of the other parameters, we chose sample problems of each group from the well-known project scheduling library (PSPLIB) [35]. As RCPSP instances in this library are only subject to the renewable resources, we used those MRCPSP instances that are subject to both renewable and nonrenewable resources. Seven sets of MRCPSP instances that differ in the related number of activities were used. Consistent with the notation used in PSPLIB, we refer to these sets of instances as j_{10}, j_{12}, j_{14}, j_{16}, j_{18}, j_{20}, and j_{30} where the last two digits in the name of each set represent the number of activities in every instance in that set. Every instance in any of these sets includes two renewable and two nonrenewable resources. We used the following method to transform each MRCPSP instance to a RCPSP-NR instance.

(i) In the selected MRCPSP instances, each nondummy activity has three execution modes. To perform the transformation, 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. In the rest of the paper we show the selected mode for each activity as m_1 and the other two modes are as m_2 and m_3, respectively.

(ii) We included different RCPSP-NR instances with one, two, or four nonrenewable resources in our sample problems. Depending on the number of nonrenewable resources, we transformed nonrenewable resource requirements of activities in MRCPSP to nonrenewable resource requirement of activities in RCPSP-NR according to Table 1. In this table, n_{jml} represents the amount of nonrenewable resource l required by activity j if the activity is executed under mode m in the MRCPSP.

(iii) In each RCPSP-NR instance, for each nonrenewable resource l, CR_l was determined equal to $\sum_{j=1}^{n+1} c_{jl}$. In order to determine CR_{lt} for each period t, we first set

Table 2: Specifications of group of instances.

Group	MRCPSP sets used	Value(s) of NC	Value(s) of $E(\eta)$	Number of problems
I	$j_{10}, j_{12}, j_{14}, j_{16}, j_{18}, j_{20}, j_{30}$	2	0.5CPL*, CPL, 2CPL	20 randomly selected instances from each set for each combination of $(E(\eta_1), E(\eta_2))$ summing up to 180 instances from each set and 1260 instances for the group
II	$j_{10}, j_{12}, j_{14}, j_{16}, j_{18}, j_{20}, j_{30}$	1, 2, 4	CPL	20 randomly selected instances from each set for each value of NC, summing up to 60 instances from each set and 420 instances for the group

*Critical path length of the project.

the value of $E(\eta_l)$, which is an average value for η_l. We then randomly generated CR_{lt} using the Poisson distribution with rate parameter of $CR_l/E(\eta_l)$. In this way the average number of periods for procurement of CR_l units of resource l was equal to $E(\eta_l)$. It is obvious that the actual value of η_l can be different from $E(\eta_l)$.

Table 2 specifies instances included in each of the two groups of sample problems.

9.2. Results Categorized Based on Instance Sets. Tables 3 and 4 show the number of instances solved to optimality within 20 seconds from each set in groups I and II, respectively. In both tables we observe that the number of solved instances within 20 seconds decreases as the number of activities in the instances increases. There are only a few negligible exceptions regarding this trend. The trend implies a higher degree of difficulty for solving the problem as the number of activities increases, which is of course consistent with expectation.

Based on both tables, PTA works better than EAA for all instance sets. Comparing the results of these algorithms with the results of the CPLEX solver shows that they are more efficient than the CPLEX solver for smaller instances, but, for larger instances, CPLEX solver seems to be more efficient. Both tables show that PTA and EAA work remarkably better than CPLEX solver for solving instances with less than 20 activities. The same situation holds for the set j20 in the Table 4, but, according to the Table 3, EAA works almost the same as CPLEX solver for this set. For instances with 30 activities, CPLEX solver works better than both of these algorithms.

Both tables show that MFSA works better than CPLEX solver for almost all instances, with the only exceptions being the instances in the sets j10 in group I and j18 in group II. Comparing this algorithm with PTA and EAA shows that MFSA cannot work as well as the two algorithms for the instances with less than 20 activities, but for the instances with 30 activities MFSA works much better than both.

According to both tables, MDAA works better than CPLEX solver for the instances with 18 activities and more. For the instances with less than 18 activities, no consistent trend is observable between MDAA and CPLEX solver results. They seem to work with almost the same efficiency. The only exception in this regard is the set j10 in group I where there is some remarkable differences between the two algorithms. According to the tables, MDAA does not work as well as PTA and EAA for the instances with 18 activities or less and it does not work as well as MFSA for the instances with 14 activities or less. But for the instances with 30 activities,

Table 3: Number of instances of each set of group I out of 180 solved to optimality in 20 seconds.

Sample problems sets	CPLEX Solver	PTA	EAA	MDAA	MFSA
j_{10}	177	180	180	137	163
j_{12}	162	180	180	165	179
j_{14}	144	180	175	131	140
j_{16}	108	177	160	111	109
j_{18}	72	166	118	76	91
j_{20}	62	125	61	64	66
j_{30}	23	16	6	23	32

Table 4: Number of instances of each set of group II out of 60 solved to optimality in 20 seconds.

Sample problems sets	CPLEX Solver	PTA	EAA	MDAA	MFSA
j_{10}	60	60	60	59	60
j_{12}	60	60	60	60	60
j_{14}	56	60	60	58	60
j_{16}	50	60	56	47	50
j_{18}	38	60	53	48	34
j_{20}	27	55	45	38	36
j_{30}	8	3	0	14	9

Table 5: Results of solving instances of group I categorized by values of parameters $E(\eta)$—Number of problems of each setting out of 140 solved to optimality in 20 seconds—Average project completion times.

$(E(\eta_1), E(\eta_2))$	CPLEX Solver	PTA	EAA	MDAA	MFSA	Average S_{n+1}
(0.5, 0.5)	127	126	122	126	123	49.3
(0.5, 1)	105	122	118	106	104	52.7
(1, 0.5)	105	117	108	100	90	46.6
(1, 1)	99	117	104	96	91	48.2
(0.5, 1.5)	70	111	87	64	75	72.8
(1.5, 0.5)	64	114	90	65	81	68.6
(1, 1.5)	58	106	85	50	64	68.6
(1.5, 1)	66	106	84	57	79	71.2
(1.5, 1.5)	53	105	82	43	73	78.6

Table 6: Results of solving instances of group II categorized by number of nonrenewable resources—Number of problems of each setting out of 140 solved to optimality in 20 seconds—Average project completion times.

Number of nonrenewable resources	CPLEX Solver	PTA	EAA	MDAA	MFSA	Average S_{n+1}
1	110	123	116	116	112	41.9
2	101	119	105	110	105	46.5
4	88	116	103	98	90	48.9

it works well comparing to the other three algorithms and remarkably better than the PTA and EAA.

9.3. Results Categorized Based on η Parameters. Table 5 shows the number of instances of group I solved to optimality by each method. In addition, the average completion times of the projects (S_{n+1}) have been presented in the last column of the table. In this table, instances are categorized based on the settings of $(E(\eta_1), E(\eta_2))$ parameters. We can observe that the number of problems solved to optimality for all algorithms and also for the CPLEX solver decreases as the values of the parameters $E(\eta_1)$ and $E(\eta_2)$ increase. This trend clearly shows the direct impact of η parameter in the computational requirement of all algorithms.

Based on the results of Table 5, the CPLEX solver works better than PTA and EAA for solving instances with small values of (0.5, 0.5) for $(E(\eta_1), E(\eta_2))$, but, for the other values of these parameters, the two algorithms work better than the CPLEX solver, and the difference turns more remarkable as the value of these parameters increases. A similar trend exists between CPLEX solver and MFSA, but the difference is that MFSA works better than CPLEX solver when the value of at least one of the parameters $E(\eta)$ is more than one. No consistent trend can be observed between MDAA and CPLEX solver. The results show that they work almost the same for most settings of $E(\eta)$ parameters.

Comparing the results of the algorithms with each other shows that PTA works better than the other three algorithms for all settings of $E(\eta)$ parameters. We can also observe that EAA works better than the other two algorithms, MDAA and MFSA, for all settings of $E(\eta)$ parameters except for (0.5, 0.5). Finally, MDAA works better than MFSA for the first four settings but not for the other settings. In all these cases, the difference between the algorithms turns more remarkable as the parameters $E(\eta_1)$ and $E(\eta_2)$ increase.

By ignoring a few exceptions, the results of the table show that, as the values of $E(\eta)$ parameters increase, the average completion time of the project increases too. This trend is of course consistent with the expectation. However, as the results show, no consistent rate of change can be found here.

9.4. Results Categorized Based on Number of Nonrenewable Resources. Table 6 shows the number of instances of group II solved to optimality by each method. In addition, the average completion times of the projects (S_{n+1}) have been presented in the last column of the table. In this table, instances are categorized based on the number of nonrenewable resources. We can observe that, as the number of nonrenewable resources increases, the effectiveness of the algorithms and the CPLEX solver decreases. This trend clearly shows the direct impact of this parameter in the degree of difficulty of each instance.

Based on the results in Table 6, for the instances with one or two nonrenewable resources, the four algorithms and the CPLEX solver can be ranked as PTA, MDAA, EAA, MFSA, and CPLEX solver in terms of their effectiveness, with PTA being the most effective algorithm and the CPLEX solver being the least effective. For the instances with four nonrenewable resources the order is slightly different and it is as PTA, EAA, MDAA, MFSA, and CPLEX solver. We also observe that the difference between the algorithms and also between them and the CPLEX solver gets more remarkable as the number of nonrenewable resources increases.

The results of the table show that, as the number of nonrenewable resources increases, the average completion time of the project increases too. This is the trend that is expected.

9.5. Durations of Solving Processes. In addition to the number of problems solved to optimality, we observed the execution time of the algorithms. Figures 2 and 3 plot the solving time of the instances of groups I and II by each algorithm and by the CPLEX solver, respectively. In every plot of both figures, related to each instance, there is a dot that its color illustrates the related solving time according to the legend. Instance sets in the plots are separated along horizontal axis. In Figure 2, the related area of each set in each plot contains nine rectangles, each of which has its own setting of $(E(\eta_1), E(\eta_2))$. In Figure 3, the related area of each set in each plot contains three rectangles. The number of nonrenewable resources increases from one rectangle to the next one as we go up along the vertical axis. Each of the rectangles in every plot of both figures shows the solving time of 20 instances. So the plots visually demonstrate the capability of the algorithms and the CPLEX solver for solving the instances with different parameters. In both figures, the plots have been ordered from the brightest to the darkest. We can observe that, in both figures, the PTA has the brightest plot and the CPLEX solver has the darkest plot. This means that the PTA has the fastest and the CPLEX solver has the slowest speed among all.

10. Conclusions

In this paper we introduced and studied RCPSP-NR problem. We customized four fundamental branch and bound methods of RCPSP for RCPSP-NR, including precedence tree algorithm (PTA), extension alternatives algorithm (EAA), minimal delaying alternatives algorithm (MDAA), and minimal forbidden sets algorithm (MFSA). We introduced several

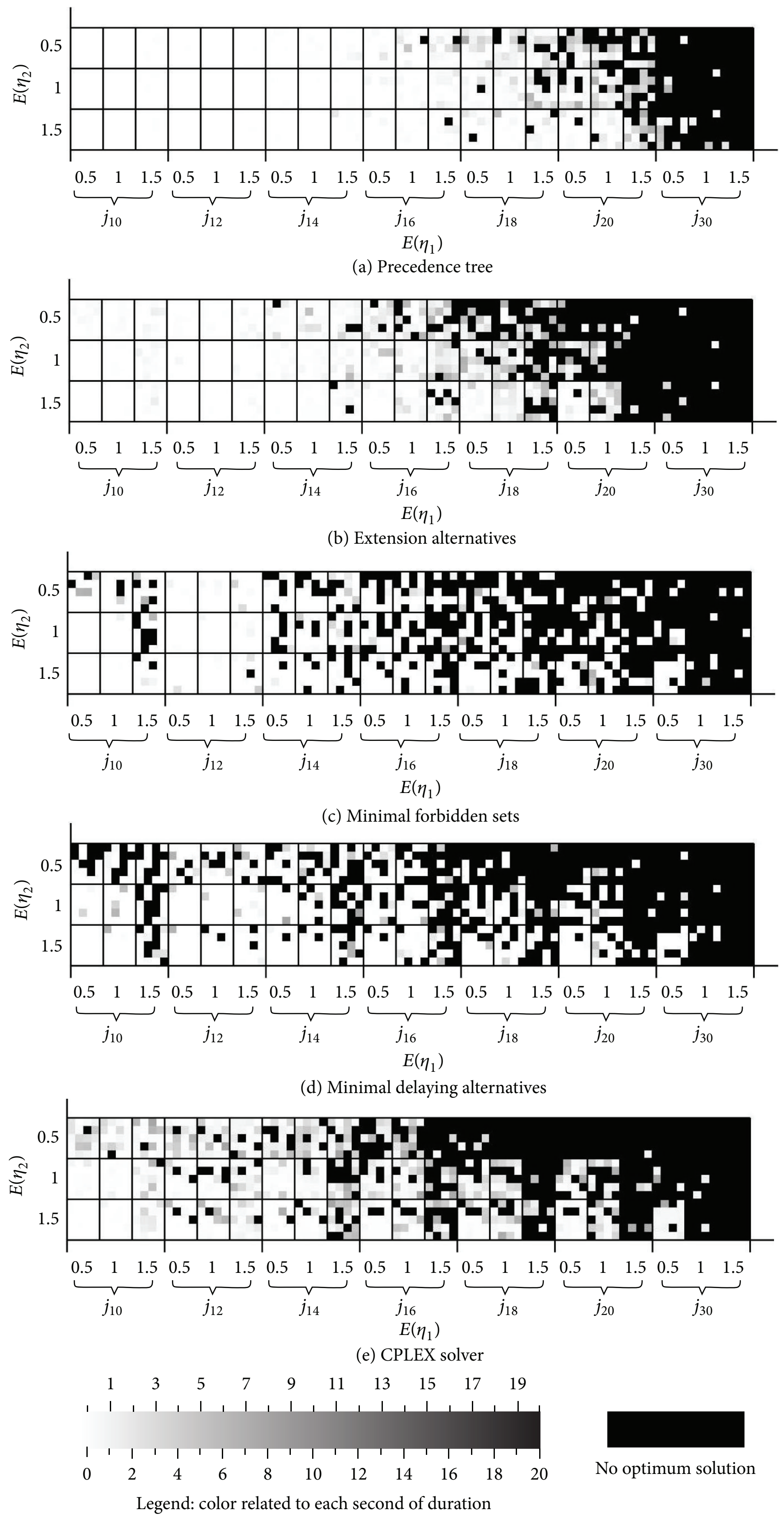
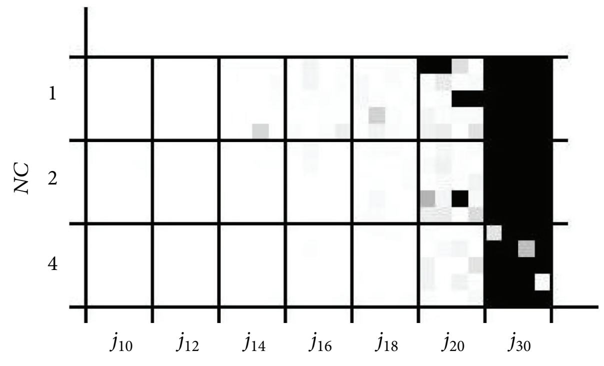
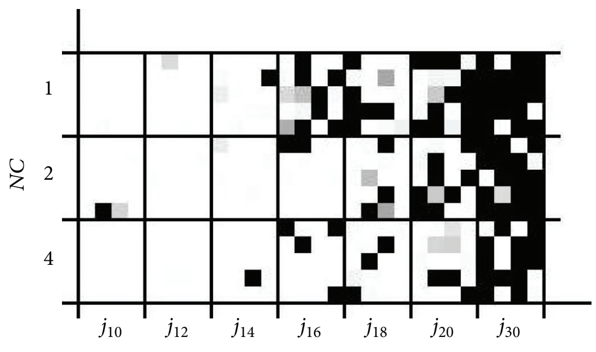


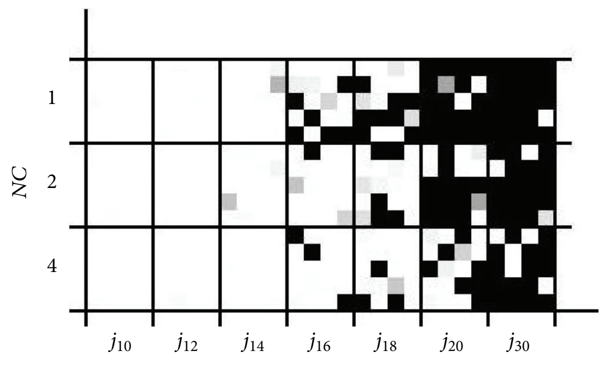
Figure 2: Solving time of instances of group I by each algorithm and also CPLEX solver.



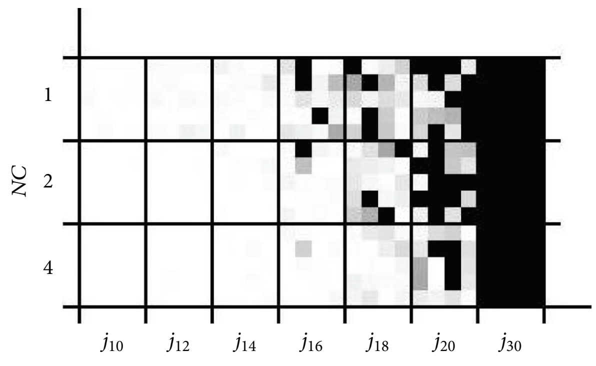
(a) Precedence tree



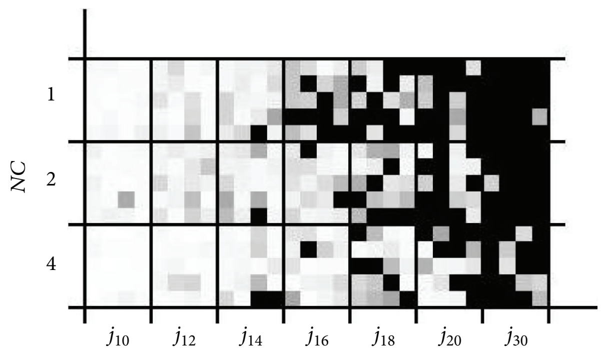
(b) Minimal delaying alternatives



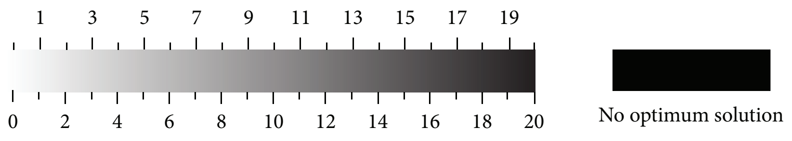
(c) Minimal forbidden sets



(d) Extension alternatives



(e) CPLEX solver



Legend: color related to each second of duration

FIGURE 3: Solving time of instances of group II by each algorithm and also CPLEX solver.

branching, bounding, fathoming, and dominance rules in the customized methods, including the ones extracted from the original methods of RCPSP and used without any change or after customization and the ones designed specifically for RCPSP-NR.

We performed a comprehensive computational experiment using all of the four proposed algorithms and reported the results. We also used CPLEX solver in the analyses and comparisons. We generated and used different instances with different numbers of activities, numbers of nonrenewable resources, and numbers of procurement periods of nonrenewable resources. Analyses showed that these three parameters affect the relative performance of the algorithms and the CPLEX solver with respect to each other. In this regard, each algorithm revealed better performance for some settings of the parameters and they generally performed better than the CPLEX solver in most of the settings.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. Böttcher, A. Drexl, R. Kolisch, and F. Salewski, "Project scheduling under partially renewable resource constraints," *Management Science*, vol. 45, no. 4, pp. 543–559, 1999.

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

[3] F. Ballestín, A. Barrios, and V. Valls, "An evolutionary algorithm for the resource-constrained project scheduling problem with minimum and maximum time lags," *Journal of Scheduling*, vol. 14, no. 4, pp. 391–406, 2011.

[4] M. Gagnon, G. Avignon, and B. Aouni, "Resource-constrained project scheduling through the goal programming model: integration of the manager's preferences," *International Transactions in Operational Research*, vol. 19, no. 4, pp. 547–565, 2012.

[5] J. Homberger, "A multi-agent system for the decentralized resource-constrained multi-project scheduling problem," *International Transactions in Operational Research*, vol. 14, no. 6, pp. 565–589, 2007.

[6] 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.

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

[8] 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.

[9] 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.

[10] J. Wglarz, 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.

[11] N. Damak, B. Jarboui, P. Siarry, and T. Loukil, "Differential evolution for solving multi-mode resource-constrained project scheduling problems," *Computers and Operations Research*, vol. 36, no. 9, pp. 2653–2659, 2009.

[12] A. Lova, P. Tormos, M. Cervantes, and F. Barber, "An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes," *International Journal of Production Economics*, vol. 117, no. 2, pp. 302–316, 2009.

[13] A. Lova, P. Tormos, and F. Barber, "Multi-mode resource constrained project scheduling: scheduling schemes, priority rules and mode selection rules," *Inteligencia Artificial*, vol. 10, no. 30, pp. 69–86, 2006.

[14] V. V. Peteghem and M. Vanhoucke, "A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 201, no. 2, pp. 409–418, 2010.

[15] L. Wang and C. Fang, "An effective estimation of distribution algorithm for the multi-mode resource-constrained project scheduling problem," *Computers and Operations Research*, vol. 39, no. 2, pp. 449–460, 2012.

[16] R. T. Harvey and J. H. Patterson, "An implicit enumeration algorithm for the time/cost trade-off problem in project network analysis," *Foundations of Control Engineering*, vol. 4, pp. 105–117, 1979.

[17] T. J. Hindelang and J. F. Muth, "A dynamic programming algorithm for decision CPM networks," *Operations Research*, vol. 27, no. 2, pp. 225–241, 1979.

[18] J. E. Kelley and M. R. Walker, "Critical-path planning and scheduling," in *Eastern Joint IRE-AIEE-ACM Computer Conference*, pp. 160–173, ACM, December 1959.

[19] C. Akkan, A. Drexl, and A. Kimms, "Network decomposition-based benchmark results for the discrete time-cost tradeoff problem," *European Journal of Operational Research*, vol. 165, no. 2, pp. 339–358, 2005.

[20] E. Demeulemeester and W. Herroelen, *Project Scheduling a research handbook*, Kluwer Academic, New York, NY, USA, 2002.

[21] A. Schirmer, *Project Scheduling with Scarce Resources*, Verlag Dr. Kovac, Hamburg, Germany, 2000.

[22] A. Schirmer and A. Drexl, "Allocation of partially renewable resources: concept, capabilities, and applications," *Networks*, vol. 37, no. 1, pp. 21–34, 2001.

[23] R. Alvarez-Valdes, E. Crespo, J. M. Tamarit, and F. Villa, "A scatter search algorithm for project scheduling under partially renewable resources," *Journal of Heuristics*, vol. 12, no. 1-2, pp. 95–113, 2006.

[24] R. Alvarez-Valdes, E. Crespo, J. M. Tamarit, and F. Villa, "GRASP and path relinking for project scheduling under partially renewable resources," *European Journal of Operational Research*, vol. 189, no. 3, pp. 1153–1170, 2008.

[25] G. Zhu, J. F. Bard, and G. Yu, "A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem," *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 377–400, 2006.

[26] A. A. B. Pritsker, L. J. Watters, and P. M. Wolfe, "Multiproject scheduling with limited resources: a zero-one programming approach," *Management Science*, vol. 16, pp. 93–107, 1969.

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

[28] C. Le Pape, "Implementation of resource constraints in Ilog Schedule: a library for the development of constraint-based scheduling systems," *Intelligent Systems Engineering*, vol. 3, no. 2, pp. 55–66, 1994.

[29] J. H. Patterson et al., "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.

[30] J. P. Stinson, E. W. Davis, and B. M. Khumawala, "Multiple resource-constrained scheduling using branch and bound," *AIIE Transactions*, vol. 10, no. 3, pp. 252–259, 1978.

[31] L. Schrage, "Solving resource-constrained network problems by implicit enumeration non-perceptive case," *Operations Research*, vol. 18, no. 2, pp. 263–278, 1969.

[32] N. Christofides, R. Alvarez-Valdes, and J. M. Tamarit, “Project scheduling with resource constraints: a branch and bound approach,” *European Journal of Operational Research*, vol. 29, no. 3, pp. 262–273, 1987.

[33] E. Demeulemeester and W. Herroelen, “A branch-and-bound procedure for the multiple resource-constrained project scheduling problems,” *Management Science*, vol. 38, no. 12, pp. 1803–1818, 1992.

[34] G. Igelmund and F. J. Radermacher, “Preselective strategies for the optimization of stochastic project networks under resource constraints,” *Networks*, vol. 13, no. 1, pp. 1–28, 1983.

[35] PSPLIB—Project Scheduling Library, http://www.om-db.wi.tum.de/psplib/main.html.