Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

An exact algorithm for minimizing resource availability costs in project scheduling

Sávio B. Rodrigues [a,*], Denise S. Yamashita [b]

[a] Department of Mathematics, Federal University of São Carlos, Caixa Postal 676, 13565-905 São Carlos, SP, Brazil
[b] Production Engineering Department, Federal University of São Carlos, Caixa Postal 676, 13565-905 São Carlos, SP, Brazil

ARTICLE INFO

Article history:
Received 30 July 2007
Accepted 1 March 2010
Available online 7 March 2010

Keywords:
Project scheduling
Resource availability
Exact algorithm

ABSTRACT

A new exact algorithm that solves the Resource Availability Cost Problem (RACP) in project scheduling is shown to yield a significant improvement over the existing algorithm in the literature. The new algorithm consists of a hybrid method where an initial feasible solution is found heuristically. The branching scheme solves a Resource-Constrained Project Scheduling Problem (RCPSP) at each node where the resources of the RACP are fixed. The knowledge of previously solved RCPSPs is used to produce cuts in the search tree. A worst-case-performance theorem is established for this new algorithm. Experiments are performed on instances adapted from the PSPLIB database. The new algorithm can be used to minimize any resource availability cost problem once a procedure for the underlying resource-constrained problem is available.

1. Introduction

We seek to minimize the total cost of renewable resources available in a project scheduling problem with a given deadline. This problem is known as the *Resource Availability Cost Problem* (RACP). The problem statement is as follows. The decision variables are the resource availabilities a_k, $k = 1, \dots, n$, and the starting times of activities s_i, $i = 1, \dots, m$. Each activity i has a duration d_i and requires r_{ik} quantities of the renewable resource k during its processing time. These activities are subject to a set H of precedence relations: when $(i,j) \in H$, activity i must precede activity j, $s_i + d_i \leqslant s_j$. The sum of resource k required by all the activities being processed at a given time must not exceed the availability a_k, for $k = 1, \dots, n$. All activities of the project must finish before or on the deadline T.

The objective is to minimize the total cost $\mathbf{c} \cdot \mathbf{a}$ where $\mathbf{a} = (a_1, \dots, a_n)$ denotes the *resource vector* and $\mathbf{c} = (c_1, \dots, c_n)$ denotes the *cost vector* with $0 < c_1 \leqslant c_2 \leqslant \dots \leqslant c_n$. This problem can be conceptually formulated as:

$$\begin{aligned} &\text{minimize} && \mathbf{c} \cdot \mathbf{a} \\ &\text{subject to:} && s_i + d_i \leqslant s_j, \quad \text{for all } (i,j) \in H, \\ &&& s_1 = 0, \\ &&& s_m \leqslant T, \\ &&& \sum_{i \in S_t} r_{ik} \leqslant a_k, \quad k = 1, \dots, n; \text{ and } t = 1, \dots, T, \end{aligned}$$

where S_t denotes the set of activities being processed at time t. Activities 1 and m are dummy activities that represent the beginning and the end of the project (they have: precedence relations $(1,i)$ and (i,m) with all the other activities, zero duration, and zero resource requirements).

An algorithm to solve the RACP was proposed in Demeulemeester (1995); we shall refer to this algorithm as the *Minimum Bounding Algorithm* (MBA). The purpose of this article is to improve the MBA. We use a hybrid method where an initial $\mathbf{a}$ is found heuristically. A branching procedure, similar to the MBA, is used to search for the optimal solution. The search space may be significantly reduced depending on the quality of the initial $\mathbf{a}$ and on the cuts that are introduced in the branching procedure. Computational experiments show an overall 24% improvement on the computational time to solve the whole instance set. We refer to the new algorithm as the *Modified Minimum Bounding Algorithm* (MMBA).

The literature on solution methods for the RACP is scant; it is also known as the *Resource Investment Problem*, Drexl and Kimms (2001). Möhring (1984) introduced the RACP motivated by a small bridge construction project. Computational experiments were reported and the author showed that the problem is NP-hard. Drexl and Kimms (2001) pointed out that the solution of the RACP for various deadlines provides time/cost tradeoffs, which is valuable information for negotiating the price for the project. Computational tests on the bridge project instances indicated that the MBA is faster than the method proposed by Möhring (1984). Rangaswamy (1998) proposed a branch-and-bound for the RACP and applied it to the same instance set used by Demeulemeester (1995). Computational results were not conclusive due to the difference in the computer platforms used in the experiments. Drexl and Kimms (2001) proposed two lower-bound procedures for the RACP based on Lagrangian relaxation and column generation methods. Moreover, feasible solutions were provided by both procedures. There are only three heuristic methods for the RACP in the literature: Yamashita et al. (2006), Shadrokh and Kianfar (2007), and Ranjbar et al. (2008).

* Corresponding author. Tel.: +55 16 3351 8220; fax: +55 16 3351 8218.
E-mail addresses: savio@dm.ufscar.br (S.B. Rodrigues), denisesy@dep.ufscar.br (D.S. Yamashita).

Closely related to the RACP is the *Resource Constrained Project Scheduling Problem* (RCPSP), which is also an NP-hard problem (Blazewicz et al., 1983). The decision variables for the RCPSP are the starting times of activities while the resource vector **a** is considered given. The objective is then to minimize the completion time of the project, i.e., minimize s_m. Note that, given **a**, the optimal schedule for the RCPSP may have a completion time that exceeds the deadline of the corresponding RACP, $s_m > T$. Thus, we consider a resource vector **a** to be *RACP-feasible* if the project can be completed before or on the RACP deadline T, otherwise we consider **a** to be *RACP-unfeasible*; for short, we say **a** is either feasible or unfeasible. Given a resource vector **a**, an algorithm that finds the optimal schedule of the RCPSP will also determine if **a** is RACP-feasible or not. A procedure that determines if **a** is RACP-feasible will be called a *subproblem solver* or simply a *solver*. In the literature there are several algorithms that solve the RCPSP; recent reviews about exact methods and heuristics can be found in Herroelen et al. (1998), Brucker et al. (1999), Kolisch and Hartmann (1998), Hartmann and Kolisch (2000, 2006).

Both algorithms, MBA and MMBA, are based on a subproblem solver. The solver we use is described in Demeulemeester (1995), where a procedure that solves RCPSP, Demeulemeester and Herroelen (1992), is modified to stop as soon as it determines if **a** is RACP-feasible or not (the RACP deadline T is used as an upper bound for RCPSP). It is worth mentioning that the MMBA and the MBA can be used as a framework to solve the RACP with different RCPSP solvers available in the literature. More generally, the MMBA and the MBA are in fact branching procedures that can be used to minimize the cost of available resources in problems other than project scheduling. They can be used whenever increasing resources also increases the feasibility of the solution and the cost function (as long as an efficient subproblem solver is available for the corresponding resource constrained problem). When the branching procedure needs to be interrupted, the MMBA yields a suboptimal solution together with a lower bound for the optimal solution while the MBA yields only a lower bound.

To generate bounds and cuts in the MMBA, the key point is to note that RACP-feasibility is directly tied to its objective function. More precisely, if a resource vector **a** is RACP-feasible, any resource vector **b** with larger or equal resources than **a** is also RACP-feasible. Equivalently, if **a** is RACP-unfeasible then any resource vector **b** with smaller or equal resources than **a** is also RACP-unfeasible. Thus, every **a** known to be RACP-unfeasible can be used to cut resource vectors from the search space. Using this idea, we show in Theorem 1 that, to verify that the incumbent feasible solution is the optimal one, a relatively small set of resource vectors in the search space needs to be verified for feasibility. Furthermore, there is a bound on the necessary number of subproblems solved in the MMBA to attain optimality in comparison to the MBA; the precise statement is found in Theorem 2.

The paper is organized as follows. Section 2 discusses how to obtain bounds for the RACP; Theorem 1 is a consequence of these bounds. Section 3 describes the algorithms MBA and MMBA. Section 4 brings an example. Section 5 contains a worst-case-performance theorem for the MMBA (Theorem 2). Section 6 shows the computational results and conclusions are drawn in Section 7.

2. Bounds

In the search for the optimal solution, various resource vectors will be examined for feasibility. Among the RACP-feasible resource vectors we seek the one with the lowest cost. At any moment in the algorithm, the *search space* is the set of resource vectors **a** for which the optimal solution may belong, i.e., excluding those that are known to be unfeasible or those that have an objective function greater than a known upper bound. We define a resource vector **b** to be *greater* than a resource vector **a**, and use the notation **b** > **a**, when $b_j > a_j$ for some j and $b_i \geqslant a_i$ for every $i \neq j$. We denote $\mathbf{b} \geqslant \mathbf{a}$ when $b_i \geqslant a_i$ for every i. Analogous definitions are used for $\mathbf{b} < \mathbf{a}$ and $\mathbf{b} \leqslant \mathbf{a}$. If we know a resource vector **b** is unfeasible then any other **a** such that $\mathbf{b} \geqslant \mathbf{a}$ is also unfeasible. To prove that a resource vector **b** is an optimal solution, perhaps a non-unique solution, we need to show that every resource vector **a** with a smaller cost function is unfeasible. In fact, we only need to show this for the resource vectors $\mathbf{a} \in S(\mathbf{b})$, where the set $S(\mathbf{b})$ is called the *slice* defined by **b**. The set $S(\mathbf{b})$ is defined as

$$S(\mathbf{b}) = \{\mathbf{a} | \mathbf{a}^{min} \leqslant \mathbf{a}, \mathbf{c}\cdot\mathbf{b} - c_1 \leqslant \mathbf{c}\cdot\mathbf{a} < \mathbf{c}\cdot\mathbf{b}\},$$

and where $\mathbf{a}^{min}$ is the resource vector with all availabilities a_k^{min} at their lower bounds $a_k^{min} = \max_i\{r_{ik}\}$. This assertion is proved in the following theorem.

Theorem 1. *Let* **b** *be RACP-feasible,* **b** *is an optimal solution if, and only if, every* $\mathbf{a} \in S(\mathbf{b})$ *is RACP-unfeasible.*

Proof. If **b** is optimal then every $\mathbf{a} \in S(\mathbf{b})$ must be unfeasible because $\mathbf{c}\cdot\mathbf{a} < \mathbf{c}\cdot\mathbf{b}$. It remains to show that if every $a \in S(\mathbf{b})$ is unfeasible then **b** is optimal. Note that for every $\mathbf{a}' \geqslant \mathbf{a}^{min}$ such that

$$\mathbf{c}\cdot\mathbf{a}' < \mathbf{c}\cdot\mathbf{b} - c_1,$$

there is an $\mathbf{a} \in S(\mathbf{b})$ which is greater than $\mathbf{a}'$. Take **a** to be the resource vector with $a_i = a'_i$ for $2 \leqslant i \leqslant n$ and

$$a_1 = \left\lceil \frac{\mathbf{c}\cdot\mathbf{b} - \sum_{i=2}^n c_i a_i}{c_1} \right\rceil - 1. \tag{1}$$

The objective function is

$$\mathbf{c}\cdot\mathbf{a} = c_1 \left\lceil \frac{\mathbf{c}\cdot\mathbf{b} - \sum_{i=2}^n c_i a_i}{c_1} \right\rceil - c_1 + \sum_{i=2}^n c_i a_i,$$

where

$$\mathbf{c}\cdot\mathbf{a} \geqslant c_1 \left(\frac{\mathbf{c}\cdot\mathbf{b} - \sum_{i=2}^n c_i a_i}{c_1} \right) - c_1 + \sum_{i=2}^n c_i a_i = \mathbf{c}\cdot\mathbf{b} - c_1,$$

and where

$$\mathbf{c}\cdot\mathbf{a} < c_1 \left(\frac{\mathbf{c}\cdot\mathbf{b} - \sum_{i=2}^n c_i a_i}{c_1} \right) + \sum_{i=2}^n c_i a_i = \mathbf{c}\cdot\mathbf{b}.$$

Thus, $\mathbf{a} \in S(\mathbf{b})$ and $\mathbf{a}' < \mathbf{a}$. Therefore, if every $\mathbf{a} \in S(\mathbf{b})$ is unfeasible then all the other resource vectors $\mathbf{a}'$, with smaller objective function, are also unfeasible. □

Remark. This theorem suggests that the dimension of the search space is reduced since the first resource is a function of the other $n - 1$ resources (formula (1)).

Now we proceed to the discussion of the search strategy.

3. The algorithm

Consider two strategies to search for the optimal solution: the *minimum bounding strategy* and the *maximum bounding strategy*. In the minimum bounding strategy the resources start at their lower bounds and are increased until the first feasible solution is found. This is the strategy used in Demeulemeester (1995) which leads to the MBA (Algorithm 2). It uses a branching scheme to increase the resources one at a time while keeping the resource vectors listed in ascending order of objective function value. In the maximum bounding strategy one starts with a RACP-feasible solution and tries to decrease the resources. Making use of Theorem 1, one would have to search for a feasible solution in the slice $S(\mathbf{b}')$ where $\mathbf{b}'$ is the incumbent solution. If a feasible solution is found in $S(\mathbf{b}')$ then the incumbent solution is improved, otherwise $\mathbf{b}'$ is optimal. Both strategies have advantages but in opposite situations. The minimum bounding strategy performs well if the optimal solution is close to the lower bound of the resources while the maximum bounding strategy performs well if the optimal solution is close to the initial feasible solution $\mathbf{b}'$. The MMBA seeks to combine the best features of both strategies. It uses the branching scheme of the minimum bounding strategy and it seeks good cuts by calling the solver to find unfeasible resource vectors in, or close to, the incumbent slice.

The MMBA uses two lists: the *list of candidate points* (LCP) and the *list of known cuts* (LKC). It has four major procedures: the *subproblem solver* (SPS), *first feasible solution* (FFS), *find cut candidate* (FCC), and *initialization of LCP* (ILCP).

The LCP stores the data for the branching scheme. It is a linked list where its jth element, LCP(j), is a resource vector $\mathbf{a}^j = (a_1^j, a_2^j, \dots, a_n^j)$; elements can be added or removed from the list. In the LCP, the resource vectors are ordered by increasing values of the objective function, $\mathbf{c}\cdot\mathbf{a}^1 \leqslant \cdots \leqslant \mathbf{c}\cdot\mathbf{a}^L$. The LCP has the property that the first element $\mathbf{a}^1$ = LCP(1) is the resource vector with the smallest objective function for which every resource vector $\mathbf{a}$, such that $\mathbf{c}\cdot\mathbf{a} < \mathbf{c}\cdot\mathbf{a}^1$, is known to be RACP-unfeasible. Thus, if $\mathbf{a}^1$ is found to be RACP-feasible then it is an RACP optimal solution. In Demeulemeester (1995) the list LCP is called the *efficient set* (ES).

The list LKC stores the resource vectors $\mathbf{u}^1, \mathbf{u}^2, \dots$ which have been found to be unfeasible.

The procedure SPS($\mathbf{a}$) is the subproblem solver; it determines if $\mathbf{a}$ is RACP-feasible or not (the same solver is used in Demeulemeester (1995) where it is called *rcps*).

The procedure FFS finds the first RACP-feasible solution $\mathbf{b}'$ using a greedy heuristic described in Yamashita et al. (2006). Starting with ample resources, the heuristic consists of decreasing the available resources by one unit, one resource at a time. The resource to be decreased is selected from the highest to the smallest resource cost. Given a resource vector, a finishing time for the project is obtained by applying a dispatch rule heuristic (Tormos and Lova, 2003). The current solution is updated if a new RACP-feasible solution is found, otherwise an attempt is made to reduce the next resource. This procedure is repeated until it is not possible to decrease the availability of any of the resources and still obtain a feasible schedule for the project using the dispatching rule.

The procedure ILCP initializes the LCP with improved lower bounds for the resource vectors. A pseudocode describing the ILCP can be found in Demeulemeester (1995) (appendix, items 1–4). An abstract description of ILCP is included here. Using SPS, ILCP finds the minimum value of A_k^{min} such that $(\infty, \dots, \infty, A_k^{min}, \infty, \dots, \infty)$ is feasible; note that, $a_k^{min} \leqslant A_k^{min}$. Now, ILCP makes various solver calls to find the lower bounds for pairs of resources a_1 and a_k, $k = 2, \dots, n$. This means that ILCP finds the smallest a_k such that $\mathbf{a} = (a_1, \infty, \dots, \infty, a_k, \infty, \dots)$ is feasible for each $a_1, A_1^{min} \leqslant a_1 \leqslant a_1^{max}$, where a_1^{max} is an upper bound for a_1; thus, a_k is a function of a_1 denoted by $a_k(a_1)$. This is done for all resources $a_k(a_1)$, $k = 2, \dots, n$. The LCP is initialized with the set of resource vectors $LCP = \{\mathbf{a}(A_1^{min}), \mathbf{a}(A_1^{min}+1), \dots, \mathbf{a}(A_1^{min}+p)\}$ where $\mathbf{a}(a_1) = (a_1, a_2(a_1), \dots, a_n(a_1))$ and $0 \leqslant p \leqslant a_1^{max} - A_1^{min}$. The resource vectors in LCP are placed in ascending ordered of objective function. We remark that ILCP may be a very time-consuming procedure. In computational tests we have observed instances where it is responsible for all but one solver call (Fig. 3). Thus, depending on the instance, ILCP is essentially responsible for solving the instance.

The procedure FCC ($\mathbf{a}, \mathbf{b}', \alpha, \mathbf{v}$) returns a vector $\mathbf{u}$ such that $\mathbf{a} \leqslant \mathbf{u}$ and $\mathbf{c}\cdot\mathbf{u} < \mathbf{c}\cdot\mathbf{b}'$ where $\mathbf{c}\cdot\mathbf{a}$ is a lower-bound and $\mathbf{c}\cdot\mathbf{b}'$ is an upper-bound for the RACP (incumbent solution). We propose a selection rule for $\mathbf{u}$ which has two parameters: a direction given by $\mathbf{v} = (v_1, \dots, v_n)$ and a *gap parameter* α, $0 \leqslant \alpha \leqslant 1$. Starting from $\mathbf{u} = \mathbf{a}$, the procedure FCC increases a resource of $\mathbf{u}$, chosen at random, by one unit at a time until the value of $\mathbf{c}\cdot\mathbf{u}$ reaches approximately, but does not exceed, the objective value $\alpha\mathbf{c}\cdot\mathbf{b}' + (1-\alpha)\mathbf{c}\cdot\mathbf{a}$. Thus, α represents the fraction of the gap between the lower- and upper-bounds where one wishes to evaluate $\mathbf{u}$. When $\alpha = 1$ the procedure returns $\mathbf{u} \in S(\mathbf{b}')$, and when $\alpha = 0$ it returns $\mathbf{u} = \mathbf{a}$. If SPS($\mathbf{u}$) finds $\mathbf{u}$ to be feasible then $\mathbf{u}$ is the new incumbent solution, and if SPS($\mathbf{u}$) finds $\mathbf{u}$ to be unfeasible then $\mathbf{u}$ is a new cut. In computational experiments we use $\alpha = 3/4$ for all instances while in the example given in Section 4 we use $\alpha = 1$. The FCC procedure is as follows:

```
Procedure FCC:
01 Initially set $\mathbf{u} := \mathbf{a}$;
02 Repeat the steps:
03   Let $k, 1 \leqslant k \leqslant n$, be the maximum index $k$ such that
04   $\mathbf{c}\cdot\mathbf{u} + c_k < \alpha\mathbf{c}\cdot\mathbf{b}' + (1-\alpha)\mathbf{c}\cdot\mathbf{a}$
05   If no such index exists, return $\mathbf{u}$ and exit procedure
06   else
07     $s := \sum_{i=1}^{k} v_i$;
08     Select one resource $j, 1 \leqslant j \leqslant k$, with probability $v_j/s$
09      Set $u_j := u_j + 1$
10   end
11 end
```

Steps 3 and 4 ensure that $\mathbf{c}\cdot\mathbf{u}$ does not exceed the desired objective value after the increment in step 9.

The MMBA, our main algorithm, is as follows:

```
**Algorithm 1**. The MMBA
01 LCP :=ILCP;
02 $\mathbf{b}'$ :=FFS;
03 $\alpha := 3/4$;
04 $\mathbf{v} := (c_1^{-1}, \dots, c_m^{-1})$;
05 $\mathbf{a}^1$ :=LCP(1);
06 Initialize LKC as empty
07 If $\mathbf{c}\cdot\mathbf{a}^1 = \mathbf{c}\cdot\mathbf{b}'$ then stop ($\mathbf{b}'$ is optimal)
08 Repeat
09   u_is_feasible :=TRUE;
10   while u_is_feasible
11     $\mathbf{u}$ :=FCC($\mathbf{a}^1, \mathbf{b}', \alpha, \mathbf{v}$);
12     u_is_feasible :=SPS($\mathbf{u}$);
13     If u_is_feasible
14       $\mathbf{b}' := \mathbf{u}$;
15       If $\mathbf{a}^1 = \mathbf{b}'$ then stop ($\mathbf{b}'$ is optimal)
16     end
17   end
18   Insert $\mathbf{u}$ in LKC;
19   branch :=TRUE;
20   while branch
21       For $k := 1, \dots, n$
22         $\mathbf{a}^{(k)} := (a_1^1, \dots, a_{k-1}^1, u_k + 1, a_{k+1}^1, \dots, a_n^1)$;
23         If $\mathbf{c}\cdot\mathbf{a}^{(k)} < \mathbf{c}\cdot\mathbf{b}'$ and there is no $\mathbf{a}$ in LCP such
           that $\mathbf{a} \leqslant \mathbf{a}^{(k)}$ then insert the vector $\mathbf{a}^{(k)}$
           in LCP (ordered by increasing cost)
24       end
25       Remove the first element of LCP;
26       If LCP is empty then stop ($\mathbf{b}'$ is optimal)
27       $\mathbf{a}^1$ :=LCP(1);
28       If $\mathbf{c}\cdot\mathbf{a}^1 \geqslant \mathbf{c}\cdot\mathbf{b}'$ then stop ($\mathbf{b}'$ is optimal)
29       If $\mathbf{a}^1 \leqslant \mathbf{u}'$ for some $\mathbf{u}'$ in LKC
30         $\mathbf{u} := \mathbf{u}'$;
31       else
32       branch :=FALSE;
33     end
34   end
35 end
```

In the above algorithm, lines 1–5 initialize the variables, lines 11–16 select a vector $\mathbf{u}$ and find if it is a new cut or a new incumbent solution, lines 21–24 add new branches to LCP, lines 25–27 set $\mathbf{a}^1$ as the next element of LCP, and line 29 verifies if a bound for $\mathbf{a}^1$ already exists. The algorithm will return $\mathbf{b}'$ as the optimal solution.

For comparison we describe the MBA algorithm (equivalent to item 5, appendix, Demeulemeester 1995).

Algorithm 2. The MBA
```
01 LCP :=ILCP;
02 $\mathbf{a}^1$:=LCP(1);
03 is_feasible :=SPS($\mathbf{a}^1$);
04 while is_feasible = FALSE
05   For $k := 1,\dots,n$
06     $\mathbf{a}^{(k)} := (a_1^1,\dots,a_{k-1}^1,a_k^1+1,a_{k+1}^1,\dots,a_n^1)$;
07     If there is no $\mathbf{a}$ in LCP such that $\mathbf{a} < \mathbf{a}^{(k)}$
       then insert the vector $\mathbf{a}^{(k)}$ in
       LCP (ordered by increasing cost)
08   end
09   Remove the first element of LCP;
10   $\mathbf{a}^1$:=LCP(1);
11   is_feasible :=SPS($\mathbf{a}^1$);
12 end (the optimal has been found.)
```

We remark that there is a gain in the branching scheme on lines 22 and 29 of the MMBA when compared to line 6 of the MBA. In the MMBA some descendants of $\mathbf{a}^1$ do not need to be examined for feasibility as they are known to be unfeasible. The performance of the MMBA depends on the choice of $\mathbf{u}$ in the FCC procedure. Among the strategies for choosing $\mathbf{u}$ such that $\mathbf{a}^1 \leqslant \mathbf{u}$, the simplest one is to take $\mathbf{u} = \mathbf{a}^1$. Doing so, one recovers the MBA algorithm. To improve cuts, it is desirable to increase the resources of $\mathbf{u}$ as in the FCC procedure where the direction $\mathbf{v}$, $\mathbf{v} > 0$, is used as a bias when increasing $\mathbf{u}$. In our computational experiments we use the direction $\mathbf{v} = \left(\frac{1}{c_1},\frac{1}{c_2},\dots,\frac{1}{c_n}\right)$. This direction $\mathbf{v}$ was chosen because $\mathbf{x} = \mathbf{a}^1 + \frac{\alpha}{n}\mathbf{c}\cdot(\mathbf{b}'-\mathbf{a}^1)\mathbf{v}$, where $\mathbf{x} = (x_1,\dots,x_n)$, maximizes the volume $\prod_{i=1}^n (x_i - a_i^1)$ subject to the constraint $\mathbf{c}\cdot\mathbf{x} = \alpha\mathbf{c}\cdot\mathbf{b}' + (1-\alpha)\mathbf{c}\cdot\mathbf{a}^1$. This will hopefully maximize the number of points to be cut from the search space.

4. Example of the algorithm

In this section, we illustrate the branching scheme of the MMBA with an example. For simplicity we assume $n = 2$ and $\alpha = 1$. Fig. 1 gives the graphical interpretation. As a reference, we list the coordinates of resource vectors (points) marked on the figure: A1 = (2,2), A2 = (6,2), A3 = (2,5), A4 = (8,2); U1 = (5,4), U2 = (7,3), U3 = (3,5), U4 = (9,2), U5 = (8,2); B0 = (8,3), B1 = (9,2).

We are assuming that the resource costs are $c_1 = 1$ and $c_2 = 2$ and that the initial heuristic solution is B0 = (8,3). Also, let us assume that the unfeasible resource vectors are below the solid polygonal line passing through B0; then the optimal solution is at B1 = (9,2). The dotted lines through B0 and B1 indicate where the cost is equal to 14 and 13, respectively. Let the lower bound for each resource be equal to 2, i.e., (a_1, a_2) is known to be unfeasible if $a_1 < 2$ or if $a_2 < 2$. In this example, we will initialize LCP = {A1} and skip the procedure ILCP. This is done because, otherwise, the simplest nontrivial example of the MMBA would require $n = 3$. When $n = 2$, ILCP alone can be used to find the optimal solution.

The important lines to be observed are line 11, where procedure FCC is called, and line 27 where $\mathbf{a}^1$ receives the first element of LCP.

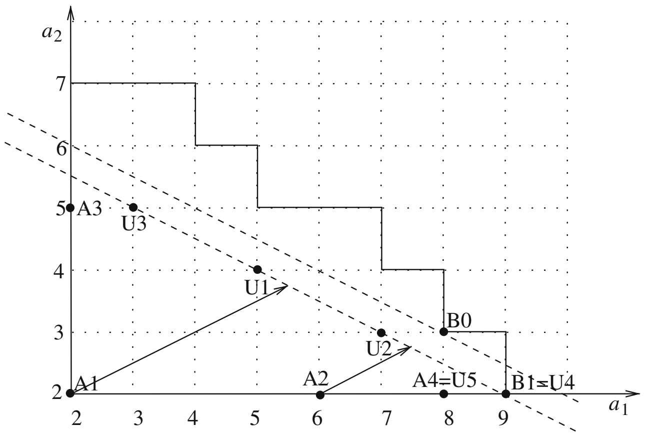

Fig. 1. Graphical example of the MMBA.

The procedure FCC has a random component; we choose a plausible outcome to be able to proceed with the example.

Starting the MMBA, the first value for $\mathbf{u}$ is obtained on line 11. Note that $\alpha = 1$, $\mathbf{a}^1 = A1$, and $\mathbf{b}' = B0$; the procedure FCC must select a resource vector $\mathbf{u}$ such that $13 \leqslant \mathbf{c}\cdot\mathbf{u} < 14$ (slice region, Theorem 1) and such that $\mathbf{u} \geqslant A1$; let us assume $\mathbf{u} = U1$ as a probable outcome (in Fig. 1 the vector with origin in A1 is a multiple of $\mathbf{v} = (1, 1/2)$). Proceeding to line 12, $\mathbf{u}$ is found to be unfeasible. Thus, all points of the search space inside a rectangle with diagonal points A1 and U1 are known to be unfeasible (12 in total). In the branching scheme, lines 21–24, the points A2 and A3 are added to LPC, line 25 removes the first element of LCP, LCP = {A2,A3}, and on line 27 we find $\mathbf{a}^1 := A2$.

Now we start the next iteration of the algorithm, on line 11 we assume that FCC returns $\mathbf{u} := U2$. On line 12, U2 is found to be unfeasible (four points are eliminated from the search space). The branching scheme adds the point A4 to the LCP list. The point (6,4) in not added to LCP since it has a cost equal to the incumbent (cost 14, see line 22). After line 25 we have LCP = {A3,A4}, and $\mathbf{a}^1 := A3$ on line 27.

More iterations are necessary. On line 11, the procedure FCC must return $\mathbf{u} := U3$; U3 is unfeasible. No new point is added to the LCP during the branching because (4,5) and (2,6) also have costs equal to the incumbent solution. Now LCP = {A4}, and $\mathbf{a}^1 := A4$ on line 27. On line 11, $\mathbf{u} := U4$ and, on line 12, U4 is found to be feasible; a new incumbent solution. The incumbent solution is updated $\mathbf{b}' := U4$ (B1 = U4). The algorithm returns to line 11 where now $\mathbf{u} := U5$ (U5 = A4). U5 is found to be unfeasible on line 12. No new points are added to LCP in the branching; the algorithm stops on line 26 because the LCP is empty. The current incumbent solution $\mathbf{b}' = B1$ is optimal.

Note that, in this example, there were five solver calls with a sequence of arguments {U1,..., U5}. There is also a sequence of vectors $\{\mathbf{a}_j^1\} = \{A1,\dots,A4\}$; and a sequence of incumbent solutions {B0, B1}. Similar sequences will be used in the next section.

5. Comparison theorem

The theorem compares the MMBA to the MBA in terms of the number of solver calls. The theorem ensures that the MMBA cannot be much worse than the MBA if a reasonable initial solution exists and α is not chosen close to 1. Note that the number of solver calls in ILCP is the same for both algorithms.

To prove the theorem, the insertion of an element into LCP must follow the same procedure in both algorithms (lines 23 in Algorithm 1, lines 7 in Algorithm 2). Moreover, this procedure should *consistently order* any pair of elements inserted in the LCP, i.e., given any two resource vectors $\mathbf{a}_1$ and $\mathbf{a}_2$, the procedure will always order $\mathbf{a}_1$ before $\mathbf{a}_2$ or it will always order $\mathbf{a}_2$ before $\mathbf{a}_1$ and this order is independent of the other elements in LCP.

Theorem 2. *The number of subproblems solved by the MMBA minus the number of subproblems solved by the MBA is at most*

$$\max\left\{0, \left\lceil \log_{1/\alpha} \frac{C_h - C^*}{(1-\alpha)c_1} \right\rceil + 1\right\} - 1,$$

where $0 < \alpha < 1$, C_h is the value of the objective function of the initial heuristic solution, and C^* is the optimal value. This result holds if both algorithms consistently order the elements of the search space in the same way.

Proof. When the SPS($\mathbf{a}$) is called, we say that $\mathbf{a}$ is *evaluated* for feasibility. The MBA evaluates the first element $\mathbf{a}^1$ of LCP at each step (Algorithm 2, lines 3 and 11); thus generating a sequence $\{\tilde{\mathbf{a}}_j^1\}$, $j = 1, \dots, N$, where $\{\tilde{\mathbf{a}}_j^1\}$, $j = 1, \dots, N-1$ are unfeasible and $\tilde{\mathbf{a}}_N^1$ is the optimal (feasible) solution. There is a total of N solver calls after ILCP.

The MMBA evaluates a vector $\mathbf{u}$ such that $\mathbf{a}^1 \leqslant \mathbf{u}$ (Algorithm 1, line 12). This defines a sequence $\{\mathbf{u}_i\}$, $i = 1, \dots, P$, along the algorithm. Let $\{\mathbf{u}_{i_j}\}$, $j = 1, \dots, Q$, denote the sub-sequence of unfeasible vectors $\mathbf{u}$ evaluated on line 12; let $\{\mathbf{u}_{i_l}\}$, $l = 1, \dots, M$, denote the sub-sequence of feasible vectors evaluated on line 12. Define $\{\mathbf{a}_j^1\}$, $j = 1, \dots, R$, to be the sequence of resource vectors that appear as the first element of LCP along the MMBA algorithm (lines 5 and 27). Note that, with consistent ordering, both algorithms will always order any two elements $\mathbf{a}_1$ and $\mathbf{a}_2$ in the LCP in the same way even if $\mathbf{c} \cdot \mathbf{a}_1 = \mathbf{c} \cdot \mathbf{a}_2$ (same tie breaker is used). In this way, if $\mathbf{a}_1$ is ordered before $\mathbf{a}_2, \mathbf{a}_2$ may appear as the first element of LCP only if $\mathbf{a}_1$ is known to be unfeasible.

Now we show that in the worst case $Q = N - 1$. Define $\mathscr{S}$ to be the set of resource vectors in the sequence $\{\tilde{\mathbf{a}}_j^1\}$, $j = 1, \dots, N$. Equivalently, $\mathscr{S} - \{\tilde{\mathbf{a}}_N^1\}$ is the set of all the elements of the initial search space that are ordered in LCP before the optimal solution $\tilde{\mathbf{a}}_N^1$. Now we show that $\mathbf{a}_j^1 \in \mathscr{S}$ for every $j = 1, \dots, R$, and thus $R \leqslant N$. Assuming there exists an $\mathbf{a}_j^1$ not in $\mathscr{S}$ leads to a contradiction: if $\mathbf{a}_j^1$ is ordered after the optimal solution $\tilde{\mathbf{a}}_N^1$, the algorithm will stop before reaching $\mathbf{a}_j^1$ because $\tilde{\mathbf{a}}_N^1$ is feasible. Since the number of unfeasible $\{\mathbf{u}_{i_j}\}$ is not greater than the number of unfeasible $\{\mathbf{a}_j^1\}$, we conclude that $Q \leqslant \min\{R, N-1\}$; $Q = N - 1$ is the worst case for the MMBA.

It remains to find the largest possible value of M yielding a total of $P = N - 1 + M$ solver calls. The MMBA yields a new incumbent solution $\mathbf{b}' = \mathbf{u}$ whenever $\mathbf{u}$ is found to be feasible in line 12. The sequence of incumbent solutions is $\{\mathbf{b}_l'\}$, $l = 0, \dots, M$, where $\mathbf{b}_l' = \mathbf{u}_{i_l}$ for $l \geqslant 1$, and where $\mathbf{b}_0'$ is obtained heuristically (assumed to require a low computational cost). From the procedure FCC, line 5, and Algorithm 1, line 11, we know that

$$\mathbf{c} \cdot \mathbf{b}_l' < \alpha \mathbf{c} \cdot \mathbf{b}_{l-1}' + (1-\alpha)\mathbf{c} \cdot \mathbf{a}_{J(l)}^1, \tag{2}$$

where $\mathbf{a}_{J(l)}^1$ is the first element in the LCP when $\mathbf{u}_{i_l}$ is evaluated. Since $\mathbf{c} \cdot \mathbf{a}_{J(l)}^1 \leqslant C^*$, it follows that

$$\mathbf{c} \cdot \mathbf{b}_l' - C^* < \alpha(\mathbf{c} \cdot \mathbf{b}_{l-1}' - C^*),$$

for $l = 1, \dots, M$. Repeated use of the above inequality gives

$$\mathbf{c} \cdot \mathbf{b}_l' - C^* < \alpha^l (C_h - C^*), \tag{3}$$

where $\mathbf{c} \cdot \mathbf{b}_0' = C_h$. Also from (2), using that $C^* \leqslant \mathbf{c} \cdot \mathbf{b}_l'$ and that $\alpha x + (1-\alpha)y = x - (1-\alpha)(x-y)$, we find

$$C^* < \mathbf{c} \cdot \mathbf{b}_{l-1}' - (1-\alpha)(\mathbf{c} \cdot \mathbf{b}_{l-1}' - \mathbf{c} \cdot \mathbf{a}_{J(l)}^1).$$

This is equivalent to

$$(1-\alpha)(\mathbf{c} \cdot \mathbf{b}_{l-1}' - \mathbf{c} \cdot \mathbf{a}_{J(l)}^1) < \mathbf{c} \cdot \mathbf{b}_{l-1}' - C^*,$$

and using (3) gives

$$\mathbf{c} \cdot \mathbf{b}_{l-1}' - \mathbf{c} \cdot \mathbf{a}_{J(l)}^1 < \frac{\alpha^{l-1}}{1-\alpha}(C_h - C^*). \tag{4}$$

If there is an index q such that $\mathbf{c} \cdot \mathbf{b}_l' - \mathbf{c} \cdot \mathbf{a}_q^1 < c_1$, for some index $q \geqslant J(l)$, then the procedure FCC will return $\mathbf{u}_{I(q)} = \mathbf{a}_q^1$ where $I(q)$ is the index in the sequence $\{\mathbf{u}_i\}$ that corresponds to $\mathbf{a}_q^1$. Since $\mathbf{c} \cdot \mathbf{b}_l'$ is a decreasing sequence and $\mathbf{c} \cdot \mathbf{a}_j^1$ is an increasing sequence, it follows that $\mathbf{c} \cdot \mathbf{b}_l' - \mathbf{c} \cdot \mathbf{a}_j^1 < c_1$ for all $j \geqslant q$. Then the equality $\mathbf{u}_{I(q)+s} = \mathbf{a}_{q+s}^1$ will be true for all $s = 0, \dots, N-q$. Thus, the next feasible $\mathbf{u}_{I(q)+s}$ will be the optimal $\mathbf{u}_P = \mathbf{b}_M'$. This implies that $M - 1$ can only be so large as to yield

$$\mathbf{c} \cdot \mathbf{b}_{M-1}' - \mathbf{c} \cdot \mathbf{a}_{J(M-1)}^1 < c_1$$

because, when this happens, the next incumbent solution must be the optimal $\mathbf{b}_M'$.

From (2) it follows that

$$\mathbf{c} \cdot \mathbf{b}_{M-1}' - \mathbf{c} \cdot \mathbf{a}_{J(M-1)}^1 < \alpha(\mathbf{c} \cdot \mathbf{b}_{M-2}' - \mathbf{c} \cdot \mathbf{a}_{J(M-1)}^1),$$

where we can apply (4) to conclude that the largest possible value for M is the smallest non-negative integer such that

$$\frac{\alpha^{M-1}}{1-\alpha}(C_h - C^*) < c_1,$$

resulting in

$$M \leqslant \max\left\{0, \left\lceil \log_{1/\alpha} \frac{C_h - C^*}{(1-\alpha)c_1} \right\rceil + 1\right\}.$$

This, together with $P = N - 1 + M$, gives the bound stated in the theorem. □

Remark. If the first $\mathbf{a}^1$ of LCP (Algorithm 1, line 5) is the optimal solution, all evaluations of $\mathbf{u}$ in line 12 are feasible and the gap between the upper and lower bounds is reduced by a factor α in every iteration.

6. Computational experiments

In this section, we evaluate the performance of the MMBA and of the MBA on 384 instances for the RACP. All computational experiments were performed on a PC Pentium 4 (2.80 GHz with 1.0 GB of RAM). All procedures were coded in C++ language following the pseudo-codes described in Section 3 and in Demeulemeester (1995). The same ILCP procedure code was used in step 1 for both the MMBA and the MBA algorithms. The test set was adapted from existing instances of the PSPLIB, which is a library for project scheduling problems. We selected instances of the RCPSP with 30 activities and four resource types and adapted them following the methodology proposed by Drexl and Kimms (2001). Two important parameters for Progen are the network complexity (*NC*) and resource factor (*RF*), Kolisch et al. (1995). *NC* reflects the average number of immediate successors of an activity. *RF* varies between 0 and 1, reflecting the density of the different resource types needed by an activity. For example, if *RF* = 1, each activity requires all types of resources, while *RF* = 0 indicates that activities do not require any type of resources. It is also necessary to determine a deadline for the project. Drexl and Kimms (2001) compute the deadline *T* for the project as a function of the project's critical path. Specifically,

$$T = DF \max_i \{EF_i\},$$

where *DF* is the deadline factor and EF_i is the earliest finishing time of activity *i*. In this paper, we generated instances with four deadline factors: 1.0, 1.2, 1.4, and 1.6. For each instance, the costs c_k were drawn from a uniform distribution $U[1,10]$. The following values for

RF and *NC* parameters were used in the PSPLIB. Resource Factor (*RF*): 0.25, 0.5, 0.75, and 1.0. Network complexity (*NC*): 1.5, 1.8, and 2.1. For each combination of *DF*, *RF*, and *NC*, eight instances are generated, resulting in a total of $4 \times 4 \times 3 \times 8 = 384$ instances. In all instances we used $\mathbf{v} = \left(\frac{1}{c_1}, \frac{1}{c_2}, \dots, \frac{1}{c_m}\right)$ and $\alpha = 3/4$. The stopping criteria for MBA and MMBA is set to 7200 seconds for each instance. The analysis of the computational data is described in two parts according to the stopping criteria. First, there are 288 instances where both algorithms find the optimum within the allowed time. Second, there are 96 instances where at least one of the algorithms fails to reach the optimum before the stopping time.

Fig. 2 shows the computational time of 288 instances where both algorithms find the optimum before the stopping time. The horizontal axis is the computational time of the MMBA while the vertical axis is the computational time of the MBA. As the time-scales can vary from 0.04 seconds to 7200 seconds, a *logarithmic scale* is used in the vertical and in the horizontal axis. The *diagonal line* indicates equal computational time for both algorithms; points above the diagonal line represent instances where the MMBA is faster than the MBA. The MMBA is faster than the MBA in many instances which require more than 10 seconds while the MMBA is slower than the MBA in many instances which require less than 3 seconds. The total computational time of these 288 instances is 33,391 seconds for the MBA and 25,376 seconds for the MMBA; an overall improvement of 24%. The MBA is faster than the MMBA in 125 instances by 2.6 seconds on average; it is faster by 142 seconds in the best instance. The MMBA is faster than the MBA in 151 instances by 55 seconds on average; it is faster by 993 seconds in the best instance (17 instances were ties). The MMBA provides a significant reduction in the computational time for instances which take between 3 and 300 seconds; it is slightly better for instances that take more than 300 seconds. We have observed that instances with larger *RF* usually require more computational time: with $RF = 0.25$ instances usually take less than 1 second but they require more than 50 seconds on average for $RF = 0.75$ and $RF = 1.00$.

The cuts introduced in the branching scheme of the MMBA (line 29 of Alg. I) are responsible for the improvement in the computational time of the MMBA. To show that these cuts indeed reduce the number of subproblems that need to be solved, Fig. 3 shows the number of SPS calls of each algorithm after ILCP. Both axis are in a logarithmic scale where they mark the number of solver calls for each algorithm during the branching scheme; the horizontal axis is the MMBA and the vertical axis is the MBA. Points above the diagonal line represent instances where the MMBA made less solver calls than the MBA. Note that the MMBA is consistently better for instances with more than 30 SPS calls, sometimes by orders of magnitude. In the bottom left hand corner, there are instances where the MBA is better than the MMBA; for example, on the horizontal axis, there are eight instances where the MBA makes only one SPS call; for these instances the number of SPS calls of the MMBA is small (as expected form the worst-case-performance theorem, Section 5).

It is possible to see the resemblance between Figs. 2 and 3; we introduced a measure to find a correlation between computational time and the number of solver calls. The *time saved* by one algorithm is the difference in computational time of the two algorithms; the time saved by one algorithm divided by the time of the slower algorithm is the *relative time saved* by the faster algorithm. The same notion can be introduced for the *relative number of solver calls* saved by an algorithm. Consider the vectors $\mathbf{x}$ and $\mathbf{y}$ defined by

$$x_i = \frac{t_i^1 - t_i^2}{\max\{t_i^1, t_i^2\}}, \quad y_i = \frac{s_i^1 - s_i^2}{\max\{s_i^1, s_i^2\}},$$

where t_i^1 is the computational time of the MMBA for instance i, t_i^2 is the computational time of the MBA, s_i^1 is the number of solver calls of the MMBA, and s_i^2 is the number of solver calls of the MBA; the vectors $\mathbf{x}$ and $\mathbf{y}$ measure the relative saving of computational time and solver calls, respectively. The correlation of these vectors is 87%; the time saved by an algorithm is strongly correlated to the saving in the number of solver calls. In this correlation, we introduced a time tolerance of 0.2 s, i.e., we excluded from $\mathbf{x}$ and $\mathbf{y}$ the instances where $|t_i^1 - t_i^2| < 0.2$.

There are instances where the optimal solution was not reached by either the MBA or the MMBA before the stopping time of 7200 seconds. The primary obstacle in the unsolved instances is the increase in computational time of the subproblem. The unsolved instances show an increase in the average computational time required in each call of SPS when compared to the average computational time required in other instances; in extreme cases, a single SPS call consumes most of the computational time. The distribution of the unsolved instances is shown in Table 1; the number of instances where the MMBA was stopped is marked on the left and the number of instances where the MBA was stopped is marked on the right. There are 90 unsolved instances where both

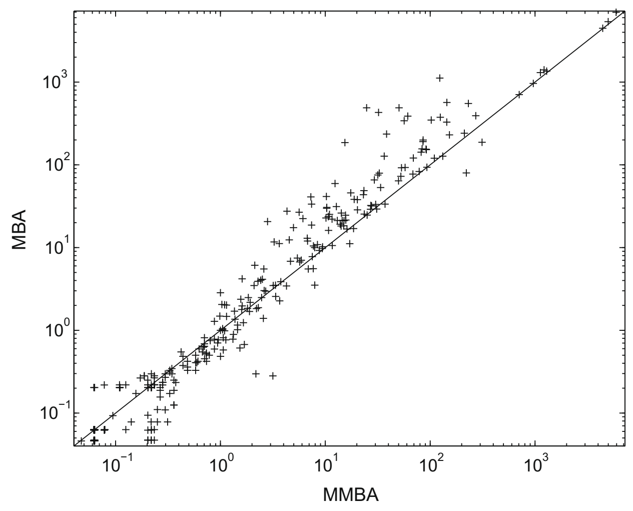

Fig. 2. Logarithmic scale of the computational time of MMBA and of the MBA, in seconds, for 288 instances solved within the stopping time.

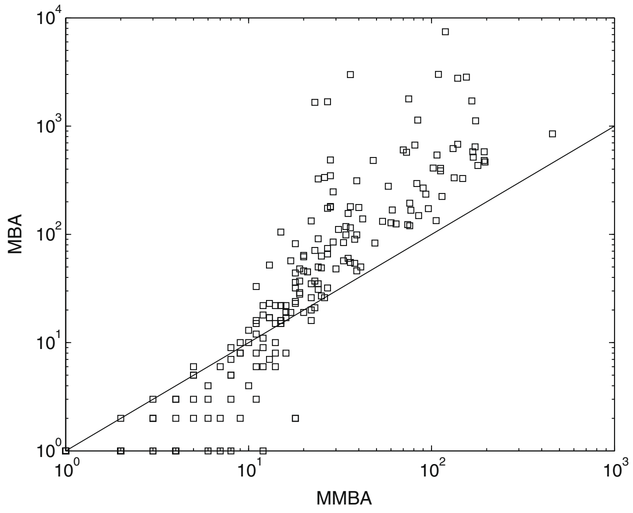

Fig. 3. Number of solver calls of MBA (line 12 of Alg. I) and MMBA (line 11 of Alg. II) for the 288 instances after the procedure ILCP.

Table 1
Number of instances, according to instance parameters, for which the optimal solution was not reached by the MMBA/MBA within the 7200 seconds stopping time. For each pair (NC,RF) there are 32 instances and for each value of RF there are 96 instances in total.

NC × RF	0.25	0.5	0.75	1.0
1.5	0/0	6/9	18/18	28/28
1.8	0/0	0/0	10/10	25/25
2.1	0/0	0/0	0/0	5/4
DF	1.0	1.2	1.4	1.6
	5/5	28/29	27/28	32/32

algorithms were stopped before reaching the optimum. In 75 instances of these 90 unsolved instances, the algorithms were interrupted inside ILCP, i.e., even before the branching schemes were started. There are four instances where the MMBA finds an optimum and the MBA does not; there are two instances where the MBA finds an optimum and the MMBA does not. There is an increase in the number of unsolved instances either when *RF* or *DF* are increased or when *NC* is decreased. Increasing *RF* clearly increases the complexity of the instance because activities become more interdependent. The subproblem becomes less restricted either when *NC* is decreased or when *DF* is increased because less restrictions are imposed on the activities. There is a sharp transition in the number of unsolved instances when *DF* changes form 1.0 to 1.2 and when NC changes form 2.1 to 1.8. All the instances solved by only one of the algorithms occur before this sharp transition: the MMBA solved three instances with $NC = 1.5$ and $RF = 0.5$, and it solved 1 instance with $NC = 2.1$ and $RF = 1.0$; the MBA solved two instances with $NC = 2.1$ and $RF = 1.0$. The most common reason why either one of the algorithms fails to reach the optimum is the existence of very time-consuming subproblems. For example, in one particular instance, the MBA evaluated five subproblems after ILCP until the stopping time was reached. For this same instance the MMBA reached the optimum in 2100 seconds with four subproblem evaluations after ILCP. Although either algorithm can fail if they encounter a set of hard subproblems, the MMBA will generally outperform the MBA when the subproblems are easy enough. For example, with $DF = 1.6$, $RF = 1.0$ and $NC = 2.1$, there is one instance where the MBA evaluates 685 subproblems after ILCP until the stopping time is reached; for this instance the MMBA reached the optimum in 4750 seconds with 66 subproblem evaluations after ILCP.

In order to access the difficulty associated with the number of activities, we experimented with 45 activities: three instances were generated for each parameter *DF*, *RF*, and *NC* a total of $3 \times 4 \times 4 \times 3 = 144$ instances. The MMBA solves 43 instances and the MBA solves one instance less than the MMBA. In 95 instances, out of the 101 that remain unsolved, both algorithms are stopped inside ILCP. Out of the 42 instances which are solved by both algorithms, 31 instances are solved with one subproblem evaluation after ILCP. Thus, there are only nine instances where the branching scheme is used to obtain the solution: in seven instances the MMBA is faster by 415 seconds on average and in two instances the MBA is faster by 1 second on average. Instances with 45 activities pose a great computational effort for the underlying RCPSP subproblem; the MMBA shows a significant improvement on the instances where the branching scheme is used.

7. Conclusion

We presented the MMBA algorithm for the exact solution of the RACP. Computational experiments showed that the MMBA is a significant improvement when compared to the existing algorithm in the literature, the MBA, especially for instances that require more computational time. This improvement was obtained by a combination of a heuristic initial solution and new bounds for the branching scheme that reduced the number of subproblem solver calls. Theorem 2 gives a bound on the number of subproblem calls of MMBA when compared to the MBA: in the worst case, the MMBA needs to solve a few more subproblems than the MBA (as observed in computational experiments). For the hardest instances, we identified the difficulty in solving the RCPSP subproblem as the cause for the increase in computational time. The instance parameters where the hardest instances are likely to occur are those with less restrictions on the starting times of the activities. We remark that the MMBA can be used as a framework to solve the RACP with different RCPSP solvers available in the literature. Moreover, it can be used in other problems that seek to minimize resource availability cost.

Acknowledgments

The authors thank the anonymous referees for their helpful comments and suggestions. This research was partially supported by FAPESP (Grant 07/00209-6).

References

Blazewicz, J., Lenstra, J.K., Rinnooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5, 11–24.
Brucker, P., Drexl, A., Mohring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: Notation, classification, models and methods. European Journal of Operational Research 112, 3–41.
Demeulemeester, E., 1995. Minimizing resource availability costs in time-limited project networks. Management Science 41, 1590–1598.
Demeulemeester, E., Herroelen, S., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38, 1803–1818.
Drexl, A., Kimms, A., 2001. Optimization guided lower and upper bounds for the resource investment problem. Journal of the Operational Research Society 52, 340–351.
Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.
Hartmann, S., Kolisch, R., 2006. Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 17, 23–37.
Herroelen, W., De Reyck, B., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers & Operations Research 25, 279–302.
Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41, 1693–1703.
Kolisch, R., Hartmann, S., 1998. Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling: Recent Models, Algorithms, and Applications. Kluwer Academic Publishers, pp. 147–178.
Möhring, R.F., 1984. Minimizing costs of resource requirements in project networks subject to a fixed completion time. Operations Research 32, 89–120.
Rangaswamy, B., 1998. Multiple Resource Planning and Allocation in Resource-Constrained Project Networks. Ph.D. Thesis, Graduate School of Business, University of Colorado.
Ranjbar, M., Kianfar, F., Shadrokh, S., 2008. Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. Applied Mathematics and Computation 196, 879888.
Shadrokh, S., Kianfar, F., 2007. A genetic algorithm for resource investment project scheduling problem, tardiness permitted with penalty. European Journal of Operational Research 181, 86–101.
Tormos, P., Lova, A., 2003. An efficient multi-pass heuristics for project scheduling with constrained resources. International Journal of Production Research 41, 1071–1086.
Yamashita, D.S., Armentano, V.A., Laguna, M., 2006. Scatter search for Project scheduling with resource availability cost. Special issue on scatter search. European Journal of Operational Research 169, 623–637.