GRASP and Path Relinking for Project Scheduling under Partially Renewable Resources

R. Alvarez-Valdes [†] , **E. Crespo** [‡] , **J.M. Tamarit** [†] , **F. Villa** [§]

[†] University of Valencia, Department of Statistics and Operations Research, Burjassot, Valencia, Spain

[‡] University of Valencia. Department of Financial Mathematics, Valencia, Spain

[§]Florida Universitaria, Valencia, Spain

[1]

Abstract

Recently, in the field of project scheduling problems the concept of partially renewable resources has been introduced. Theoretically, it is a generalization of both renewable and non-renewable resources. From an applied point of view, partially renewable resources allow us to model a large variety of situations that do not fit in classical models, but can be found in real problems in timetabling and labor scheduling. In this paper we develop some preprocessing techniques and several heuristic algorithms for the problem. Preprocessing significantly reduces the dimension of the problems, therefore improving the efficiency of solution procedures. Heuristic algorithms based on GRASP and Path relinking are then developed and tested on existing test instances, obtaining excellent results.

Keywords: Project management and scheduling; Partially renewable resources; Heuristics; GRASP; Path relinking

1 Introduction

Project scheduling consists of allocating scarce resources to the set of activities in a project over time. Project scheduling has been the object of a great deal of research since the first methods, CPM[7] and PERT[15], were developed in the 1950's. These initial methods were able to manage large projects and were considered a useful tool in the planning process. However, they assumed unlimited resources, an assumption severely reducing their application to most real problems. Therefore, many researchers started to study the resource-constrained case (RCPSP). Up to now, many exact and heuristic algorithms have been developed (see the book by Demeulemeester and Herroelen[4] for an excellent state-of-the -art description). The now classic RCPSP basically includes two types of resources: renewable resources, in which the availability of each resource is renewed at each period of the planning interval, and non-renewable resources, whose availability is given once for the whole project and are consumed throughout the process of the activities requiring them.

[1]Email addresses: ramon.alvarez@uv.es, enric.crespo@uv.es, jose.tamarit@uv.es, fvilla@florida-uni.es

However, in recent years new types of resource have been proposed to allow the model to include new types of constraints. An example is allocatable resources[9, 14], in which the units of the resource required by a given operation i remain occupied from the start of a given allocating activity up to the completion of activity i. This type of resource is useful to model situations in which the units of a resource are not indistinguishable. For instance, if the resource corresponds to workers of a given type, the person performing an activity consisting of a direct service to clients must carry out some other activity involving the same clients and cannot be changed for another worker of the same type.

Another example of new resources is cumulative resources[10, 11]. In this case, the amount of a resource decreases when it is used by some activities, but it can increase as the result of the process of some other activities, as is the case in some industries involving chemical products.

Another new type of resource is partially renewable resources. The availability of the resource is associated to a subset of periods of the planning horizon and the activities requiring the resource only consume it if they are processed in these periods. Although these resources may seem strange at first glance, they can be a powerful tool for solving project scheduling problems. On the one hand, from a theoretical point of view, they include renewable and non-renewable resources as particular cases. In fact, a renewable resource can be considered a partially renewable resource with an associated subset of periods consisting of exactly one period. Non-renewable resources are partially renewable resources where the associated subset is the whole planning horizon. On the other hand, partially renewable resources make it possible to model complicated labor regulations and timetabling constraints, therefore allowing us to approach many labor scheduling and timetabling problems as special cases of project scheduling problems.

As an example, let us consider a project involving human resources. We can find some contractual conditions like that of *working at most 2 weekend days out of every 3 consecutive weeks*. This condition cannot be modelled as a renewable resource, because this type of resource considers each period separately. It cannot be modelled as a non-renewable resource because this type of resource considers the whole planning horizon. We model this condition as a partially renewable resource with a set of periods $\{6, 7, 13, 14, 20, 21\}$ for the first three weekends and a total availability of 2 units. Each task consumes 1 unit of this resource for each weekend day in which it is processed. In Figure 1 we see three activities A, B, and C scheduled within the timescale depicted above. Activity A is in process at periods 6 and 7 and then it consumes 2 units of the resource. Activity B does not consume the resource and activity C consumes 1 unit in period 20. If these 3 activities had to be done by the same worker, the solution in the figure would not be possible because it would exceed resource availability.

Partially renewable resources were first introduced by Böttcher et al.[1] in 1999. They proposed an integer formulation and developed exact and heuristic algorithms. Schirmer[13] studied these new type of resources thoroughly in his book on project scheduling problems. He presented many examples of special conditions which can be suitably modelled using partially renewable resources. He also proposed several families

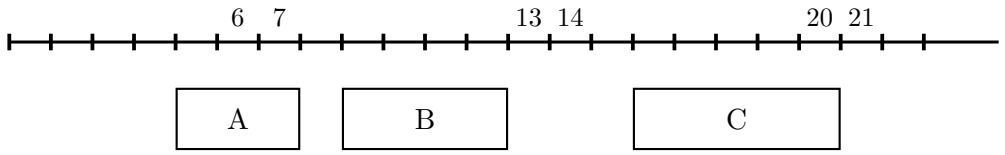

Figure 1: *Example of partially renewable resource*

of approximate algorithms for solving the RCPSP/π.

In this paper we develop some preprocessing techniques and several heuristic algorithms for project scheduling under partially renewable resources. Preprocessing reduces the dimension of the problems in terms of resources and possible finishing times for the activities in the project, therefore improving the efficiency of the algorithms. Some heuristic algorithms, based on GRASP and Path Relinking, are then developed and tested on existing test instances. In Section 2 the elements of the problem are defined and an integer formulation provided. Section 3 contains the preprocessing routines. In Section 4 we develop the heuristic algorithms. Section 5 is devoted to the computational experience and Section 6 to conclusions and future lines of research.

2 Formulation of the problem

The RCPSP/π can be defined as follows: Let J be the set of $n = |J|$ activities, numbered from 1 to n, where activity 1 and activity n are dummy activities representing the beginning and end of the project. Let P_j be the set of activities which are immediate predecessors of activity j and P'_j the set of all predecessors of j. Each activity j has a duration of d_j and cannot be interrupted. Let R be the set of partially renewable resources. Each resource $r \in R$ has a total availability K_r and an associated set of periods Π_r. An activity j requiring resource r will consume k_{jr} units of it at each period $t \in \Pi_r$ in which it is processed. Finally, let T be the planning horizon in which all the activities must be processed. For each activity j we obtain the earliest and latest finishing times, EFT_j, LFT_j, by critical path analysis. We denote $E_j = \{EFT_j, ..., LFT_j\}$, the set of possible finishing times, and $Q_{jt} = \{t, ..., t + d_j - 1\}$.

The RCPSP/π consists of sequencing the activities so that the precedence and resource constraints are satisfied and the makespan is minimized.

If we define the variables:

$$x_{jt} = \begin{cases} 1 & \text{if activity } j \text{ finishes at time } t \\ 0 & \text{otherwise.} \end{cases}$$

the problem can be formulated as follows:

$$Min \quad \sum_{t \in E_n} t x_{nt} \tag{1}$$

$$s.t. \quad \sum_{t \in E_j} x_{jt} = 1 \qquad j \in J \tag{2}$$

$$\sum_{t \in E_i} t x_{it} \leq \sum_{t \in E_j} (t - d_j) x_{jt} \qquad j \in J,\ i \in P_j \tag{3}$$

$$\sum_{j \in J} k_{jr} \sum_{t \in \Pi_r} \sum_{q \in Q_{jt} \bigcap E_j} x_{jq} \leq K_r \qquad r \in R \tag{4}$$

$$x_{jt} \in \{0, 1\} \qquad j \in J,\ t \in E_j \tag{5}$$

The objective function (1) minimizes the finishing time of the last activity and hence the makespan of the project. According to constraints (2) each activity must finish once. Constraints (3) are the precedence constraints and constraints (4) the resource constraints. Note that in this problem there is only one global constraint for each resource $r \in R$. Another special characteristic of this problem is that all the activities must finish inside a closed interval E_j, because sets Π_r are defined with respect to the planning horizon T. Therefore, the existence of feasible solutions is not guaranteed. In fact, Schirmer[13] has shown that the feasibility variant of the RCPSP/π is NP-complete in the strong sense.

The above formulation is called the normalized formulation by Böttcher et al. [1] and Schimer[13]. Alternative formulations are considered in their papers, but they finally adopt the normalized formulation due to its simplicity.

3 Preprocessing

Preprocessing has two objectives. First, to decide whether a given instance is unfeasible or if it has feasible solutions. If the latter is the case, a second objective is to reduce the number of possible finishing times of the activities and the number of resources. If these two objectives are satisfactorily achieved, the solution procedures will not waste time trying to solve unfeasible problems and will concentrate their efforts on the relevant elements of the problem.

Preprocessing consists of several phases:

1. *Reducing the planning horizon T*

   For each instance, we are given a planning horizon T. This value plays an important role in the problem formulation. In fact, late finishing times of the activities, LFT_j are calculated starting from T in a backward recursion. Therefore, the lower the value T, the fewer variables the problem will have. In order to reduce T, we try to build a feasible solution for the given instance, using the GRASP

algorithm which will be described later. The GRASP iterative process stops as soon as a feasible solution is obtained or after 200 iterations. The new value T is updated to the makespan of the feasible solution obtained. Otherwise, T is unchanged.

If the makespan of the solution equals the length of the critical path in the precedence graph, the solution is optimal and the process stops and returns the solution.

2. *Eliminating idle resources*

Each resource $r \in R$ is consumed only if the activities requiring it are processed in periods $t \in \Pi_r$. Each activity can only be processed in a finite interval. It is therefore possible that no activity requiring the resource can be processed in any period of Π_r. In this case, the resource is idle and can be eliminated. More precisely, if we denote the possible processing times of activity j by $PPT_j = \{EFT_j - d_j + 1, .., EFT_j, ..., LFT_j\}$, and $\forall j \in J \mid k_{rj} > 0 : \ \Pi_r \bigcap PPT_j = \emptyset$, the resource $r \in R$ is idle and can be eliminated.

3. *Eliminating non-scarce resources*

Schirmer[13] distinguishes between scarce and non-scarce resources. He considers a resource $r \in R$ as scarce if $\sum_{j \in J} k_{jr} d_j > K_r$, that is, if an upper bound on the maximum resource consumption exceeds resource availability. In this case, the upper bound is computed by supposing that all the activities requiring the resource are processed completely inside Π_r.

We have refined this idea by taking into account the precedence constraints. Specifically, we calculate an upper bound on the maximal consumption of resource r by solving the following linear problem:

$$Max \quad \sum_{j \in J} k_{jr} \sum_{t \in \Pi_r} \sum_{q \in Q_{jt} \bigcap E_j} x_{jq} \tag{6}$$

$$s.t. \qquad \sum_{t \in E_j} x_{jt} = 1 \qquad\qquad j \in J \tag{7}$$

$$\sum_{m=t}^{T} x_{im} + \sum_{s=1}^{t+d_j-1} x_{js} \leq 1 \qquad j \in J,\ i \in P_j,\ t \leq T \tag{8}$$

$$x_{jt} \geq 0 \qquad\qquad j \in J\,, t \in E_j \tag{9}$$

The objective function (6) maximizes the resource consumption over the whole project. Constraints (7) ensure that each activity finishes once. Constraints (8) are the precedence constraints. We use this expression, introduced by Christofides et al.[3], because it is more efficient than the usual precedence constraint. In fact, this linear problem has the integrality condition and its optimal solution is always

integer [2]. If the solution value is not greater than the resource availability, this resource will not cause any conflict and can be skipped in the solution process.

4. *A filter for variables based on resources*

   For each activity j and each possible finishing time $t \in E_j$ we perform the following test to decide if this time t is not feasible for activity j to finish in. $Q'_{jrt} = \{t - d_j + 1, t - d_j + 2, ..., t\} \bigcap \Pi_r$ is the set of processing times of j which lie inside Π_r if it finishes at time t, and $m_{jrt} = |Q'_{jrt}|$ denotes the corresponding number of periods.

   - For each predecessor $i \in P'_j$, let ER_{it} be the reduced set of possible finishing times of i if j finishes at time t.
   - For each successor l ($j \in P'_l$), let ER_{lt} be the reduced set of finishing times of l if j finishes at time t.
   - Let U_j be the set of activities not related to j by precedence constraints.
   - Consider the resources $r \in R$, one at a time, and compute the minimal consumption of the resource if activity j finishes at time t:

     $$MC_r = m_{jrt}k_{jr} + \sum_{i \in P'_j} min_{s \in ER_i}\{m_{irs}k_{ir}\} + \quad \text{(predecessors of } j\text{)}$$
     $$\sum_{l|j \in P'_l} min_{s \in ER_l}\{m_{lrs}k_{lr}\} + \quad \text{(successors of } j\text{)}$$
     $$\sum_{h \in U_j} min_{s \in E_h}\{m_{hrs}|k_{hr}\} \quad \text{(remaining activities)}$$

   - If $MC_r > K_r$, time t is not feasible for activity j to finish in and the corresponding variable $x_{jt} = 0$.

   When this filter is applied to an activity j some of its possible finishing times can be eliminated. From then on, the set of possible finishing times is no longer E_j. We denote by PFT_j the set of finishing times passing the filter.

   This filter is applied iteratively. After a first run on every activity and every finishing time, if some of the variables are eliminated the process starts again but this time computing MC_r on the sets PFT_j, PFT_i, PFT_l, PFT_s instead of the original E_j, E_i, E_l, E_s. As the minimum are calculated over restricted subsets, it is possible that new finishing times fail the test and are eliminated. The process is repeated until no finishing time is eliminated in a complete run.

5. *Consistency test for finishing times*

   When the above filter eliminates a finishing time of an activity j, it is possible that some of the finishing times of its predecessors and successors are no longer feasible. For instance, suppose an activity j with $PFT_j = \{9, 10, 11, 12, 13, 14, 15\}$ and duration $d_j = 3$ and an activity $i \in P_j$ with $PFT_i = \{6, 7, 8, 9, 10, 11, 12\}$. If the resource filter eliminates $t = 15$ from PFT_j, then $t = 12$ is not possible

for activity i because if i finishes at time 12, j must necessarily finish at time 15, which is unfeasible. Therefore, time 12 for activity i is eliminated.

In general, for an activity j let us denote by $\tau_j = max\{t \,|\, t \in PFT_j\}$. Then, for each $i \in P_j$ the finishing times $t \in PFT_i$ such that $t > \tau_j - d_j$ can be eliminated. Analogously, if for activity j, $\gamma_j = min\{t \,|\, t \in PFT_j\}$, for each $i \,|\, j \in P_i$ the finishing times $t \in PFT_i$ such that $t < \gamma_j + d_i$ can be eliminated.

This test is also applied iteratively until no more finishing times are eliminated. If, after applying these two procedures for reducing variables, an activity j has $PFT_j = \emptyset$, the problem is unfeasible and the procedure stops, returning the unfeasibility status of the given instance.

6. *Constructing a trial solution*

In the first step of the preprocessing procedure we tried to build a feasible solution. If the feasible solution was obtained, we checked if its makespan was equal to the length of the critical path. If this was the case, the solution was optimal. After the elimination of variables, we then check if the makespan of that solution equals the minimum time in PFT_n. If this is the case, the solution is optimal.

Otherwise, we build a trial solution by assigning a finishing time $t_j = min\{t \,|\, t \in PFT_j\}$ to each activity j. Obviously this solution satisfies the precedence constraints. If it satisfies the resource constraints as well, it is the optimal solution.

4 GRASP algorithm

GRASP, greedy randomized adaptive search procedure, is an iterative process combining a constructive phase and an improvement phase. The construction phase builds a solution step by step, adding elements to a partial solution. The element to add is selected according to a greedy function which is dynamically adapted as the solution is built. However, the selection is not deterministic, but subjected to a randomization process. Hence, when we repeat the process we can obtain different solutions. When a feasible solution has been built, its neighborhood is explored in a local search phase until a local optimum is found. Resende and Ribeiro[12] present a comprehensive review of GRASP and an extensive survey of GRASP literature can be found in Festa and Resende[6].

4.1 The constructive phase

A deterministic constructive algorithm

We have adapted the Serial Scheduling Scheme (SSS) proposed by Schirmer[13], which in turn is an adaptation of the Serial Scheduling Scheme commonly used for the classical RCPSP. We denote by FT_j the finishing time assigned to activity j. At each stage of the iterative procedure an activity is scheduled by choosing

from among the current set of decisions, pairs (j,t) of an activity j and a possible finishing time $t \in PFT_j$. The selection is based on a priority rule.

Step 0. Initialization

$s = 1$ (counter of stage)

$FT_1 = 0$ (sequencing dummy activity 1)

$S_1 = \{1\}$ (partial schedule at stage 1)

$\forall r \in R : RK_{r1} = K_r$ (remaining capacity of resource r at stage 1)

$$TD_{r1} = \sum_{j \in J} k_{jr} d_j \text{ (maximum possible demand for } r \text{ at stage 1)}$$

$SR_1 = \{r \in R \mid TD_{r1} > RK_{r1}\}$ (set of possible scarce resources)

EL_1= set of eligible activities, those activities for which activity 1 is the only predecessor

Step 1. Constructing the set of decisions

$D_s = \{(j,t) \mid j \in EL_s\,,\, t \in PFT_j\}$

Step 2. Choosing the decision

Select the best decision (j^*, t^*) in D_s, according to a priority rule

Step 3. Feasibility test

If (j^*, t^*) is resource-feasible, go to Step 4.

Else

$D_s = D_s \setminus \{(j^*, t^*)\}$

If $D_s = \emptyset$, STOP. The algorithm does not find feasible solution.

Else, go to Step 2.

Step 4. Update

$s = s + 1$

$FT_{j^*} = t^*$

$S_s = S_{s-1} \cup \{j^*\}$

$EL_s = (EL_{s-1} \setminus \{j^*\}) \cup \{j \in J \mid P_j \subseteq S_s\}$

$\forall l \in J \mid j \in P_l : PFT_l = PFT_l \setminus \{\tau \mid t^* + d_l > \tau\}$

$\forall r \in R : \quad RK_{rs} = RK_{r,s-1} - k_{j^*r} m_{j^*rt^*}$

$$TD_{rs} = TD_{r,s-1} - k_{j^*r} d_{j^*}$$

If $TD_{rs} \leq RK_{rs}$, then $SR_s = SR_{s-1} \setminus \{r\}$

If $s = n$, STOP. The sequence is completed.

Else, go to Step 2.

At Step 1, the construction of D_s could have included the feasibility test of Step 3, as in Schirmer's[13] original scheme. However, we have preferred not to check the resource availability of every decision and only check the decision already chosen. In problems with a large number of possible finishing times for the activities, this strategy is more efficient.

We keep the set of possible scarce resources SR_s updated because some priority rules based on resource consumption only take this type of resources into account.

Priority rules

We have tested the 32 priority rules used by Schirmer[13]. The first 8 are based on the network structure, including classical rules such as EFT, LFT, SPT or MINSLK. The other 24 rules are based on resource utilization. 12 of them use all the resources and the other 12 only the scarce resources. A preliminary computational experience, which will be fully described in Section 6, allowed us to choose the most promising rules and use them in the next phases of the algorithm's development. These preliminary results also showed that even with the best performing rules the deterministic constructive algorithm failed to obtain a feasible solution for many instances of 10 activities generated by Böttcher et al.[1]. Therefore, the objective of the randomization procedures which were included in the algorithm was not only to produce diverse solutions but to ensure that for most of the problems the algorithm would obtain a feasible solution.

Randomization strategies

We introduce randomization procedures for selecting the decision at Step 2 of the constructive algorithm. Let s_{jt} be the score of decision (j,t) on the priority rule we are using and $s_{max} = max\{s_{jt}|(j,t) \in D_s\}$, and let δ be a parameter to be determined $(0 < \delta < 1)$. We have considered three alternatives:

(a) *Random selection on the Restricted Candidate List, S*
Select decision (j^*, t^*) at random in set $S = \{(j,t) \mid s_{jt} \geq \delta s_{max}\}$

(b) *Biased selection on the Restricted Candidate List, S*
The decisions involving the same activity j are given a weight which is inversely proportional to the order of their finishing times. For instance, if in S we have decisions $(2,4),(2,5),(2,7),(2,8)$ involving activity 2 and ordered by increasing finishing times, then decision $(2,4)$ will have a weight of 1, decision $(2,5)$ weight $1/2$, decision $(2,7)$ weight $1/3$ and decision $(2,8)$ weight $1/4$. The same procedure is applied to the decisions corresponding to the other activities. Therefore, the decisions in S corresponding to the lowest finishing times of the involved activities will be equally likely and the randomized selection process will favor them.

(c) *Biased selection on the set of decisions D_n*
We have also implemented the Modified Regret-Based Biased Random Sampling (MRBRS/δ) proposed by Schirmer[13], in which the decision (j,t) is chosen from among the whole set D_n but with its probability proportional to its regret value. The regret value is a measure of the worst possible consequence that might result from selecting another decision.

A repairing mechanism

The randomization strategies described above significantly improve the ability of the constructive algorithm to find feasible solutions for tightly constrained instances. However, a limited computational experience showed that not even with the best priority rule and the best randomization procedure could the constructive algorithm obtain feasible solutions for all the instances of 10 activities generated by Böttcher et al.[1]. Therefore, we felt that the algorithm was not well-prepared for solving larger problems and we decided to include a repairing mechanism for unfeasible partial schedules.

In the construction process, if at Step 3 all decisions in D_n fail the feasibility test and finally D_n becomes empty, instead of stopping the process and starting a new iteration, we try to re-assign some of the already sequenced activities to other finishing times in order to free some resources that could be used for the first of the unscheduled activities to be processed. If this procedure succeeds, the constructive process continues. Otherwise, it stops. A detailed description of the repairing mechanism is not provided because it is very similar to the double move described in the next subsection.

4.2 The improvement phase

Given a feasible solution obtained in the constructive phase, the improvement phase basically consists of two steps. First, identifying the activities whose finishing times must be reduced in order to have a new solution with the shortest makespan. These activities are labelled as *critical*. Second, moving critical activities in such a way that the resulting sequence is feasible according to precedence and resource constraints. We have designed two types of moves: simple and double. In a simple move, only a critical activity is moved, leaving the remaining activities unchanged. In a double move, non-critical activities are moved to make the move of a critical activity possible.

1. **Building M, the set of critical activities**

   Step 0. Initialization

   $M = \{n\}$ (the last activity of the project n is always critical)

   $s_n = 1$ (activity n has not yet been studied for enlarging M).

   Step 1. Adding activities to M

   While($\exists j \in M \mid s_j = 1$) {

   Take the largest $j \in M$ with $s_j = 1$

   Set $s_j = 0$

   $\forall i \in P_j$:

   If $FT_i + d_j = FT_j$ (there is no slack between i and j)

   $M = M \cup \{i\}$

   $s_i = 1$ }

At Step 1, the condition for including an activity in M simply says that if j has to be moved to the left, reducing its finishing time, a predecessor i which is processed immediately before j must also be moved to the left in order to leave room for moving j. This condition can be refined if we take into account that the preprocessing filters may have eliminated some possible finishing times of the activities. If $t'_j = max\{t \in PFT_j \,|\, t'_j < FT_j\}$, the condition of Step 1 can be written as: If $FT_i + d_j > t'_j$, then i is critical.

For instance, suppose we have activity $4 \in M$ with $FT_4 = 14$, $d_4 = 5$, $PFT_4 = \{10, 11, 12, 14\}$, and activity 2 is a predecessor of 4 with $FT_2 = 8$. If activity 4 has to be moved to the left, its new finishing time will be 12 at most and therefore FT_2 can no longer be 8. Activity 2 must be moved to the left and hence $2 \in M$. In this example, $t'_4 = 12$ and $FT_2 + d_4 = 13$.

2. **Simple move**

We try to move every activity $j \in M$ to the left, in topological order, to a new finishing time satisfying the precedence and resource constraints. If an activity cannot be moved, the procedure stops. If for an activity there are several possible new finishing times, that with minimum global resource consumption is chosen.

Step 0. Initialization

$RK_r, \forall r \in R,$ are the resources not used in the current sequence

$u_j = 1, \forall j \in M$ (activity still to be moved)

$possible = true$ (the move is still possible)

Step 1. Moving activities in M

While($\exists j \in M \,|\, u_j = 1$)

{

Take the minimum $j \in M$ with $u_j = 1$

Set $u_j = 0$

$MINFT_j = max\,\{FT_i + d_j \mid i \in P_j\}$

$t_{best} = FT_j$

$max_{excess} = 0$

$\forall t \in PFT_j \mid MINFT_j \leq t < FT_j$

{ $\forall r \in R : RK_r = RK_r + k_{jr} m_{jrFT_j} - k_{jr} m_{jrt}$

If $RK_r \geq 0, \forall r \in R$ (possible move)

$excess = \sum_{r \in R} RK_r$

If $max_{excess} < excess$

$max_{excess} = excess$

$t_{best} = t$

Recover previous RK_r }

If $t_{best} = FT_j$ (no change)

Recover the original $FT_j, \forall j \in J$ and return $false$

Else, $FT_j = t_{best}$

}

Step 2.

Return $true$ and the modified solution

3. **Double move**

Step 0. Initialization

$RK_r, \forall r \in R,$ are the resources not used in the current sequence

$u_j = 1, \forall j \in M$ (activity still to be moved)

$possible = true$ (the move is still possible)

While($\exists j \in M \mid s_j = 1$)

{

Step 1. Selecting an activity $j \in M$ to be moved

Take the minimum $j \in M$, with $u_j = 1$

Set $u_j = 0$

$MINFT_j = max\{FT_i + d_j \mid i \in P_j\}$

Step 2. Considering a new finishing time for j

$\forall t \in PFT_j \mid MINFT_j \leq t < FT_j$

{

$repaired = true$

$\forall r \in R : RK_r = RK_r + k_{jr}m_{jrFT_j} - k_{jr}m_{jrt}$

If $RK_r \geq 0, \ \forall r \in R$

$FT_j = t$

Go to Step 1, to move another critical activity

Else

Step 3. Moving other activities $i \in J$

$LC = \emptyset$, list of possible changes

$\forall i \in J \mid i \neq j$

{

$MINFT_i = max\{FT_l + d_i \mid l \in P_i\}$

$MAXFT_i = min\{FT_k - d_k \mid i \in P_k\}$

$t_{best} = FT_i$

$best_{viol} = 0$

Step 4. New finishing times for activity i

$\forall u \in PFT_i \mid MINFT_i \leq u \leq MAXFT_i$

{

$new_{viol} = 0$

{ $\forall r \in R :\ change_r = k_{ir}m_{iru} - k_{ir}m_{irFT_i}$

If $RK_r > 0$ and $RK_r - change_r < 0$

$new_{viol} = new_{viol} + (change_r - RK_r)$

$repaired = false$

If $RK_r < 0$

If $change_r < 0$

$new_{viol} = new_{viol} - min\{-RK_r, -change_r\}$

Else, $new_{viol} = new_{viol} + change_r$

If $RK_r < change_r$

$repaired = false$ }

If $repaired = true$

$t_{best} = u$

Go to Step 5

If $best_{viol} > new_{viol}$

$best_{viol} = new_{viol}$

$t_{best} = u$

} *(end of Step 4)*

Step 5. Add to the list of possible changes

If $t_{best} \neq FT_i$

$LC = LC \cup \{(i, t_{best})\}$

Update RK_r and FT_i

If $repaired = true$

Go to Step 6

} *(end of Step 3)*

Step 6. Make changes associated to activity j

If $repaired = true$

$FT_j = t$

Make changes in LC and update FT_i, RK_r

Else, return $false$

} *(end of Step 2)*

} *(end of main While)*

Step 7.

Return $true$ and the modified solution

In Step 2, the new finishing time which is being considered for activity $j \in M$ may be resource-feasible and no other activity needs to be moved. If this is not the case, in Step 3 other activities are considered for moving. An activity i is moved to a new provisional finishing time if this move offsets the resource violation provoked by moving j or, at least, reduces the deficit. Therefore, throughout the search in J, a provisional list of changes LC is built until the solution is repaired or J is exhausted. If the solution is repaired with the list of changes in LC, those moves are made and a new $j \in M$ is considered. Otherwise, the procedure stops without improving the solution.

The double move can be enhanced in the following way. If we arrive at Step 6 without completely covering the deficit created by moving j, but this deficit is partially reduced, we can go back to Step 3 and search J again from the beginning, trying to further reduce or eliminate the remaining deficit. The procedure is more complex but sometimes offers feasible moves for critical activities.

The three procedures of the improvement phase are run iteratively:

S= current solution

$improve = false$

do{

Build set M of critical activities

$improve$=SimpleMove(S, M)

if $improve = false$

$improve$=DoubleMove(S, M)

} while ($improve = true$)

4.3 An aggressive procedure

The standard version of our heuristic algorithm starts by applying the preprocessing procedure of Section 3. The reduced problem then goes through the iterative GRASP algorithm described above, combining a constructive phase and an improvement phase at each iteration, until the stopping criterion, here a fixed number of iterations, is met.

An enhanced version of the heuristic algorithm combines preprocessing and GRASP procedures in a more *aggressive* way. After a given number of iterations (stopping criterion), we check if the best known solution has improved. If this is the case, we run the preprocessing procedures again, setting the planning horizon T to the makespan of the best known solution and running the filters for variable reduction. The GRASP algorithm is then applied on the reduced problem. Obtaining feasible solutions is now harder, but if the procedure succeeds we will get high quality solutions. A scheme of the modified algorithm appears in Figure 2.

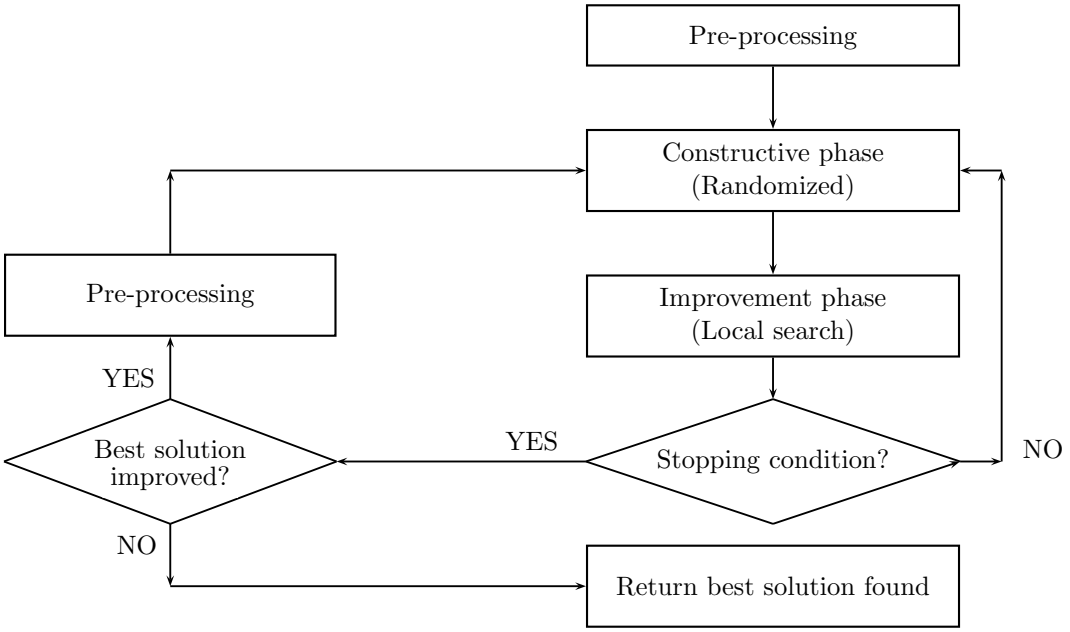

Figure 2: *Scheme of Aggressive GRASP*

4.4 Path Relinking

If throughout the iterative procedures described above we keep a set of the best solutions, usually denoted as *elite solutions*, we can perform a Path Relinking procedure. Starting from one of these elite solutions, called the *initiating solution*, we build a path towards another elite solution, called the *guiding solution*. To the intermediate solutions in the path we progressively impose the attributes of the guiding solution, so these intermediate solutions evolve from the initiating solution until they reach the guiding solution. Hopefully, along these paths we will find solutions which are better than both extremes, the initiating and the guiding solutions.

We keep the 10 best solutions obtained in the GRASP procedure. We consider one of them in turn as the initiating solution and another as the guiding solution. We

build a path from the initiating to the final solution with $n-1$ intermediate solutions. The j^{th} solution will have the finishing times of the first j activities taken from the guiding solution, while the remaining $n-j$ finishing times will still correspond to those of the initiating solution. Therefore, along the path, the intermediate solutions will be progressively more similar to the guiding solution and more different from the initiating one. In some cases these intermediate solutions will not be feasible. If this is the case, a repairing mechanism similar to that described in Section 4 is applied. We proceed from activity 1 to activity n, checking for each activity j if the partial solution from 1 to j is feasible. If it is not, we first try to find a feasible finishing time for activity j, keeping previous activities unchanged. If that is not possible, we try to re-assign some of the previous activities to other finishing times in order to obtain some resources which are necessary for processing activity j at one of its possible finishing times. If this procedure succeeds, we consider activity $j+1$. Otherwise, the solution is discarded and we proceed to the next intermediate solution. If we obtain a complete intermediate solution which is feasible, we apply to it the improvement phase described in the GRASP algorithm.

5 Computational results

5.1 Test instances

Böttcher et al.[1] generated a first set of test instances. Taking as their starting point PROGEN 2 [8], an instance generator for the classical RCPSP with renewable resources, they modified and enlarged the set of parameters and generated a set of 2160 instances with 10 non-dummy activities, 10 replications for each one of the 216 combinations of parameter values. As most of the problems were unfeasible, they restricted the parameter values to the 25 most promising combinations and generated 250 instances of sizes 15, 20, 25, 30 and 60 of non-dummy activities, always keeping the number of resources to 30.

More recently, Schirmer[13] has developed PROGEN 3, an extension of PROGEN 2, and has generated some new test instances. He has generated 960 instances of sizes 10, 20 30 and 40, with 30 resources. Most of them have a feasible solution, while a few of them are unfeasible and some of them are labelled as undecided because a time-limited run of the branch and bound algorithm by Böttcher et al.[1] failed to obtain a feasible solution. Table 1 shows the status of Schirmer's problems as reported in [13]

5.2 Preprocessing results

The preprocessing procedures in Section 3 have been applied to the Böttcher et al.[1] problems of 10, 15, 20, 25 and 30 which are available upon request from the authors. Different aspects of the results appear in Tables 2, 3 and 4. Table 2 shows the performance of preprocessing in determining problem status.

The last line of Table 2 shows the status we have been able to determine for the problems left undecided by the preprocessing procedures. We have tried to solve these instances with CPLEX, using an integer programming formulation of the problem adapted

Instance Set	Non-optimally solved	Optimally solved	Feasibly solved	Undecided	Proven infeasible	Total
J10	39	901	940	11	9	960
J20	203	734	937	23	0	960
J30	181	757	938	22	0	960
J40	183	743	926	34	0	960
Total	606	3135	3741	90	9	3840

Table 1: *Test problems generated by Schirmer*

	n=10	n=15	n=20	n=25	n=30
Problems	2160	250	250	250	250
Detected as impossible	1205	16	17	12	8
Detected as possible	879	233	231	236	239
Undecided	76	1	2	2	3
Actual status	Impossible	Possible	Undecided	Impossible	Undecided

Table 2: *Böttcher et al. problems - Determining the status*

from that appearing in Section 2, though for 2 instances of size 20 and 3 instances of size 30 long time runs of this powerful code failed to obtain even a feasible integer solution. In summary, we can say that our preprocessing procedures are very efficient in determining the actual status of a given instance.

Table 3 shows the optimal solutions that the preprocessing obtains either by proving that the initial feasible solution is optimal, or by building a trial solution in which each activity is assigned to its minimum finishing time after the reduction filters have been applied. For more than 70 % of the instances the optimal solutions are found.

	n=10	n=15	n=20	n=25	n=30
Problems	2160	250	250	250	250
Feasible problems	879	234	233	236	242
Solved to optimality by pre-processing	646	165	177	190	193
Remaining problems	233	67	56	46	49

Table 3: *Böttcher et al. problems - Optimal solutions identified in the preprocessing*

Table 4 presents the reduction in the number of resources and variables for the problems not solved in preprocessing, for which some other algorithm has to be applied. The fast preprocessing techniques significatively reduce the number of resources to be taken into account and, more importantly, the number of possible values of the decision variables.

Similar results have been obtained for the test problems generated by Schirmer[13].

	n=10	n=15	n=20	n=25	n=30
Problems	233	67	56	46	49
Initial resources	30	30	30	30	30
Remaining resources (on average)	18	18	23	25	25
Initial variables (on average)	90	268	565	874	1314
Remaining variables (on average)	51	130	348	611	906

Table 4: *Böttcher et al. problems - Reductions of resources and variables*

Table 5 shows the performance of preprocessing, first determining the status of the all of the problems and then providing optimal solutions for many of them. Note that the status of all problems left undecided in Schirmer's book[13] have been determined. In fact, all of them have been proven to be feasible, except for five instances of size 10 which are impossible. For more than 75 % of the feasible problems, the preprocessing procedures are able to provide a proven optimal solution.

	n=10	n=20	n=30	n=40
Problems	951	960	960	960
Feasible problems	946	960	960	960
Solved to optimality by pre-processing	609	727	796	793
Remaining problems	337	233	164	137

Table 5: *Schirmer problems - Optimal solutions identified in the preprocessing*

A characteristic of PROGEN 3 is that it tends to produce large values of the planning horizon T. On the one hand, that favors the existence of feasible solutions. On the other hand, as the number of possible finishing times of activities depends directly on T, a very large number of variables are initially defined. Therefore, for this set of problems the reduction of T described in Section 3 is especially useful . Table 6 shows the reduction of T obtained by that procedure on the non-optimally solved problems.

	n=10 337 problems	n=20 233 problems	n=30 164 problems	n=40 137 problems
Average initial T	43	82	120	158
Average reduction	9 (21%)	31 (38%)	52 (43%)	80 (50%)
Maximal reduction	30 (70%)	61 (76%)	100 (84%)	134 (85%)

Table 6: *Schirmer problems - Reductions of planning horizon T*

The reductions of the planning horizon T, together with the procedures for reducing possible finishing times for the activities, produce dramatic decreases in the final number of variables to be used by solution procedures. Table 7 presents the reductions in the number of resources and variables obtained by the preprocessing strategies.

	n=10 337 problems	n=20 233 problems	n=30 164 problems	n=40 137 problems
Initial resources	30	30	30	30
Remaining resources (average)	15 (50%)	15 (50%)	18 (60%)	16 (53%)
Initial variables (average)	210	965	2287	4255
Remaining variables (average)	101 (48%)	332 (34%)	720 (31%)	1062 (25%)

Table 7: *Schirmer problems - Reductions of resources and variables*

5.3 Computational results of constructive algorithms

The 32 priority rules described by Schirmer[13] were coded and embedded in the constructive algorithm of Section 4.1. These rules were tested on the 879 feasible instances of size 10 generated by Böttcher et al.[1]. Table 8 shows the results obtained by the 6 best performing rules. The first 3 rules are based on the network structure of the problems. The last 3 rules are based on resource consumption. In them, ES indicates that the rules require the use of only scarce resources, indexed by r. Rk_{rs} is the remaining capacity of resource r at stage s, as defined in Section 4.1. RD_{jrt} is the relevant demand, defined as $RD_{jrt} = k_{jr}|Q_{jt} \cap \Pi_r|$. MDE_{jrt} is the minimum relevant demand entailed for resource r by all successors of activity j when started at period t. The most important feature of Table 8 is that even the best rules fail to produce a feasible solution for 20% of these small instances of size 10. Therefore, we need randomizing strategies and repairing mechanisms to significantly increase the probability of finding feasible solutions in the constructive phase of the GRASP algorithm.

Rule	Definition	Feasible solutions (%)	Optimal solutions (%)
LFT	$Min\{LFT_j\}$	80.09	64.28
MTS	$Max\{\|\{i\|j \in P_i^{'}\}\|\}$	79.64	69.98
SLK	$Min\{LST_j - EFT_j\}$	76.22	61.66
DRC/ES	$Max\{\sum_r (RK_{rs} - RD_{jrt})\}$	81.57	27.08
DRS/ES	$Min\{\sum_r (RK_{rs}/RD_{jrt})\}$	79.29	27.53
TRS/ES	$Min\{\sum_r (RD_{jrt} + MDE_{jrt})\}$	79.41	28.56

Table 8: *Results of priority rules*

Table 9 presents the improvement in the number of feasible and optimal solutions obtained by the constructive algorithm when one of the randomizing strategies are included in Step 2. As in Table 8, the test problems are the size 10 instances of Böttcher et al.[1]. Only two rules have been kept for this second test, LFT, which is the best rule among those based on network structure and DRC/ES, the best rule based on resource usage. Table 9 shows that the randomization procedures allow us to get an important increase in the number of feasible solutions. However, not all these small problems can be solved. That is the reason for the development of a repairing mechanism to help the constructive algorithm to find feasible solutions for the more tightly constrained

problems.

Rule	Randomizing strategy	Feasible solutions (%)	Optimal solutions (%)
LFT	Deterministic	80.09	64.28
	Random 1	97.95	61.89
	Random 2	97.61	93.83
	Random 3	98.41	61.66
DRC/ES	Deterministic	81.57	27.08
	Random 1	96.25	72.81
	Random 2	95.56	76.11
	Random 3	98.41	54.38

Table 9: *Results of randomizing strategies*

Table 10 shows the final results of the complete constructive algorithm, including the repairing mechanism. From Table 9 we have kept *Random 3* because it obtains the highest number of feasible solutions and *Random 2* because it obtains the highest number of optimal solutions. The results show that the constructive algorithm now seems to be well-prepared for solving larger problems. The priority rule LFT produces many more optimal solutions than DRC/ES. This rule, based on the use of resources, is more orientated to attaining feasibility by choosing times with low resource requirements than to get optimality by processing activities as early as possible. However, as the feasibility of the solutions is guaranteed by the joint effort of a randomizing strategy and the repairing mechanism, rule LFT will be chosen for the GRASP algorithm.

Rule	Strategy	Iterations	Feasible solutions (%)	Optimal solutions (%)
LFT	Random 2	1000	99.89	99.09
	Random 2	2000	100	99.43
	Random 3	1000	100	93.63
	Random 3	2000	100	96.36
DRC/ES	Random 2	1000	99.66	89.31
	Random 2	2000	99.66	89.31
	Random 3	1000	100	81.91
	Random 3	2000	100	84.41

Table 10: *Results of the complete constructive algorithm*

5.4 Computational results of GRASP algorithms

Tables 11 and 12 show the results of the GRASP algorithms on the problems of Böttcher et al.[1] and Schirmer[13] respectively. Four versions of the algorithm have been tested: *GRASP*, the basic GRASP algorithm, *GR+PR*, in which the best solutions obtained in the GRASP iterations go through the Path Relinking phase described in Section 4.4, *AG-GR*, the modified GRASP procedure described in Section 4.3, and *AG-GR+PR*, combining modified GRASP and Path Relinking. The GRASP algorithms

use priority rule LFT and the second randomization procedure with $\delta = 0.85$. For each problem size the Tables show the number of non-optimal solutions, the average distance to optimum and the maximal distance to optimum. However, not all the optimal solutions are known. In fact, in Table 11 for 1 instance of size 20, 7 instances of size 25 and 7 instances of size 30 the optimal solution is unknown. Analogously, in Table 12 the optimal solution is not known for 1 instance of size 30 and 5 instances of size 40. In these cases, which are marked (*), the comparison is made with the best-known solution, obtained by a time-limited run of the CPLEX integer code or by heuristic methods.

Problem size	Feasible instances		$GRASP$	$GR+PR$	$AG-GR$	$AG-GR+PR$
10	879	Non-optimal	1	1	2	2
		Mean dist. (%)	0.006	0.006	0.15	0.15
		Max dist. (%)	5.6	5.6	7.7	7.7
15	234	Non-optimal	4	4	3	3
		Mean dist. (%)	0.13	0.13	0.09	0.09
		Max dist. (%)	17.9	17.9	17.9	17.9
20	231	*Non-optimal**	8	8	8	8
		Mean dist. (%)	0.41	0.41	0.33	0.33
		Max dist. (%)	24.2	24.2	27.3	27.3
25	236	*Non-optimal**	7	6	8	7
		Mean dist. (%)	0.20	0.19	0.24	0.23
		Max dist. (%)	21.7	21.7	21.7	21.7
30	239	*Non-optimal**	5	5	5	4
		Mean dist. (%)	0.10	0.10	0.06	0.05
		Max dist. (%)	11.5	5.8	3.9	3.9

Table 11: *Results of GRASP algorithms on Böttcher et al. problems*

The results in Table 11 show that only a few very difficult problems of every size are not optimally solved. However, these problems are so hard that almost no difference between algorithms can be observed. The maximum distance to optimum can be relatively very high. For instance, problem $P2408$ of size 15 has an optimal solution of 28 while the heuristic solution is 33. However, due to the special type of resources involved, it is possible that no feasible solutions of length 29, 30, 31 and 32 exist. If that were the case, only one possibility of improving is left, though the high value of the maximum distance would seem to suggest the opposite.

The results in Table 12 allow us to observe the different performance of the four algorithms more clearly. The aggressive GRASP procedure does not guarantee a better solution than the basic GRASP algorithm, as can be seen in the first row of the Table, but for larger problems it tends to produce better results. The Path Relinking algorithm adds little improvement to the good results obtained by GRASP procedures.

Tables 13 and 14 complement the information in previous Tables by providing the running times of the algorithms on both sets of problems. In all cases preprocessing is

Problem size	Feasible instances		$GRASP$	$GR+PR$	$AG-GR$	$AG-GR+PR$
10	946	Non-optimal	1	1	2	2
		Mean dist. (%)	0.003	0.003	0.007	0.007
		Max dist. (%)	2.9	2.9	3.4	3.4
20	960	Non-optimal	33	22	20	19
		Mean dist. (%)	0.12	0.08	0.07	0.06
		Max dist. (%)	13.0	13.0	13.0	13.0
30	960	*Non-optimal**	58	55	34	34
		Mean dist. (%)	0.22	0.20	0.12	0.11
		Max dist. (%)	12.1	12.1	13.6	13.6
40	960	*Non-optimal**	79	76	56	50
		Mean dist. (%)	0.48	0.42	0.25	0.22
		Max dist. (%)	32.0	32.0	20.5	20.5

Table 12: *Results of GRASP algorithms on Schirmer problems*

included as a part of the solution procedure. The algorithms have been coded in *C++* and run on a Pentium IV at 2.8 Ghz. The basic GRASP algorithm stops after 2000 iterations, while the stopping criterion of the aggressive GRASP is set to 500 iterations. The average running times are very short, though some problems would require quite long times. Adding the Path Relinking procedure increases the running times very slightly and therefore it seems convenient to keep it in the final implementation. If we compare the running times of the basic and the aggressive GRASP procedures, we do not see large differences, except in the last line of Table 14. However, that is the case in which the results of the aggressive GRASP are more clearly superior to the basic algorithm and the larger computing time is efficiently used to obtain better results. Therefore, the aggressive GRASP algorithm with Path Relinking seems to be the best option for an efficient heuristic algorithm.

Problem size		$GRASP$	$GR+PR$	$AG-GR$	$AG-GR+PR$
10	Average time	0.41	0.41	0.21	0.21
	Maximum time	30.4	30.5	45.8	45.9
15	Average time	1.34	1.35	1.25	1.25
	Maximum time	51.0	51.1	93.0	93.1
20	Average time	4.91	4.97	2.98	3.02
	Maximum time	180.8	186.6	154.9	155.2
25	Average time	8.85	8.91	6.73	6.76
	Maximum time	316.5	316.6	299.2	299.5
30	Average time	8.11	8.12	8.99	9.00
	Maximum time	455.6	455.6	457.7	457.8

Table 13: *Running times of GRASP algorithms on Böttcher et al. problems*

Problem size		$GRASP$	$GR+PR$	$AG-GR$	$AG-GR+PR$
10	Average time	0.89	0.90	1.05	1.05
	Maximum time	41.9	42.2	33.1	33.3
20	Average time	0.95	0.97	0.70	0.71
	Maximum time	162.4	163.0	53.0	53.1
30	Average time	2.03	2.10	2.04	2.11
	Maximum time	135.0	143.7	144.5	144.7
40	Average time	3.96	4.05	4.32	4.40
	Maximum time	155.6	155.8	507.5	519.8

Table 14: *Running times of GRASP algorithms on Schirmer problems*

6 Conclusions

We have studied a generalization of the classical resource constrained project scheduling problem. A new type of resource is considered, the partially renewable resource in which the availability of the resource is associated to a given set of periods and the activities only consume it when they are processed in these periods. These resources can be seen as a generalization of renewable and non-renewable resources, but their main interest comes from their usefulness to model complex situations appearing in timetabling and labor scheduling problems, which can be approached as project scheduling problems.

We have developed several preprocessing techniques which help to determine the existence of feasible solutions and to reduce the number of variables and constraints. We have also designed and implemented heuristic algorithms based on GRASP and Path Relinking. Preprocessing procedures and heuristic algorithms have been tested on two sets of instances previously proposed in the literature. They have been able to determine the feasibility status of many instances which up to now were undecided and to solve most of the feasible instances optimally.

We are convinced that the preprocessing techniques developed here should be used by any solution procedure, exact or heuristic, applied to this problem. Our heuristic algorithms are also very efficient and can be considered a useful tool for obtaining high quality solutions for the problem.

Future lines of research will be the development of an exact algorithm and the design of new heuristic algorithms for problems in which partially renewable resources are combined with classical renewable resources, as happens in real situations.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Science and Technology DPI2002-02553, and the Valencian Science and Technology Agency, GRU-

POS03/174.

References

[1] J. Böttcher, A. Drexl, R. Kolish, F. Salewski, Project Scheduling Under Partially Renewable Resource Constraints, Management Science 45 (1999) 544-559.

[2] S. Chaudhuri, R.A. Walker, J.E. Mitchell, Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 2 (1994) 456-471.

[3] N. Christofides, R. Alvarez-Valdes, J.M. Tamarit, Project scheduling with resource constraints: a branch and bound approach, European Journal of Operational Research 29 (1987) 262-273.

[4] E.L. Demeulemeester, W.S. Herroelen, Project Scheduling: A Research Handbook, Kluwer Academic Publishers, Boston, 2002.

[5] A. Drexl, R. Nissen, J.H. Patterson, F. Salewski, ProGen/πx - An instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions, European Journal of Operational Research 125 (2000) 59-72.

[6] P. Festa, M.G.C. Resende, GRASP: An annotated bibliography, in: M.G.C. Resende, P. Hansen (Eds.), Essays and Surveys in Metaheuristics, Kluwer Academic Press, Boston, 2001, pp. 325-367.

[7] J.E. Kelley, Critical path planning and scheduling: Mathematical basis, Operations Research 9 (1961) 296-320.

[8] R.Kolish, A. Sprecher, A. Drexl, Characterization and generation of a general class of resource-constrained project scheduling problems, Management Science 41 (1995) 1693-1703.

[9] C. Mellentien, C. Schwindt, N. Trautmann, Scheduling the factory pick-up of new cars, OR Spectrum (2004), in press.

[10] K. Neumann, C. Schwindt, N. Trautmann, Advanced production scheduling for batch plants in process industries, OR Spectrum 24 (2002) 251-279.

[11] K. Neumann, C. Schwindt, N. Trautmann, Scheduling of continuous and discontinuous material flows with intermediate storage restrictions, European Journal of Operational Research (2004), in press.

[12] M.G.C. Resende, C.C. Ribeiro, Greedy Randomized Adaptive Search Procedures, in: F. Glover, G. Kochenberger (Eds.), State-of-the-art Handbook in Meteheuristics, Kluwer Academic Press, Boston, 2001, pp. 219-250.

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

[14] C. Schwindt, N. Trautmann, Scheduling the production of rolling ingots: industrial context, model and solution method, International Transactions in Operations Research 10 (2000) 547-563.

[15] J.D. Wiest, F.K. Levy, A management guide to PERT/CPM, Prentice-Hall, Englewood Cliffs, New Jersey, 1976.