A Scatter Search Algorithm 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 Mathematics for Economics and Business, Valencia, Spain

[§]Florida Universitaria, Valencia, Spain

[1]

Abstract

In this paper we develop a heuristic algorithm, based on Scatter Search, for project scheduling problems under partially renewable resources. This new type of resource can be viewed as a generalization of renewable and non-renewable resources, and is very helpful in modelling conditions that do no fit into classical models, but which appear in real timetabling and labor scheduling problems. The Scatter Search algorithm is tested on existing test instances and obtains the best results known so far.

Keywords: Project management and scheduling; Partially renewable resources; Heuristics; Scatter Search

1 Introduction

Project scheduling consists of determining the starting and finishing times of the set of activities in a project. These activities are linked by precedence relations and their processing requires one or more resources. If the availability of these resources can be considered unlimited, the problem is satisfactorily solved by PERT-CPM[8, 17] techniques, in use from the early 1950's. However, most real problems involve severely constrained resources. In this case, we talk of Resource-Constrained Project Scheduling Problems (RCPSP). 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 horizon, and non-renewable resources, whose availability are given at the beginning of the project and which are consumed throughout the processing of the activities requiring them (see the book by Demeulemeester and Herroelen[5] for a state-of-the-art description of RCPSP models and algorithms). Nevertheless, these basic types of resources cannot accommodate many real situations and some other types of resources have been proposed, as in the case of allocatable resources[11, 16] and cumulative resources[12, 13].

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

In this paper we consider partially renewable resources, another new type of resource introduced by Böttcher et al.[2] in 1999. The availability of a partially renewable resource is associated to a subset of periods in the planning horizon and the activities requiring the resource only consume it if they are processed within these periods. This type of resource 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 as a set of partially renewable resources, each one 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.

Böttcher et al.[2] proposed an integer formulation and developed exact and heuristic algorithms. Schirmer[15] studied this 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 of approximate algorithms for solving the RCPSP/π. Alvarez-Valdes et al.[1] developed GRASP and Path Relinking algorithms.

In this paper we describe some preprocessing techniques and develop a new heuristic algorithm 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. A heuristic algorithm, based on Scatter Search, is 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 Scatter Search algorithm. 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 dummy activities 1 and n represent 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 last period of 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 re-

source 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. Constraints (2) ensure that each activity finishes once. Constraints (3) are the precedence constraints and constraints (4) the resource constraints. Note that unlike the problem with renewable resources in which there is a constraint for each resource and each period, 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 inside the planning horizon $(0, T)$. Therefore, the existence of feasible solutions is not guaranteed. In fact, Schirmer[15] 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. [2] and Schimer[15]. 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, helping 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.

The preprocessing we have developed includes several procedures:

Identifying trivial problems

If the solution in which the finishing time of each activity i is set to EFT_i is resource-feasible, then it is optimal.

Reducing the planning horizon

For each instance, we are given a planning horizon $(0, T)$. This value T plays an important role in the problem formulation. The latest 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 briefly 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. If no feasible solution is found, T is unchanged.

If the makespan of the solution is equal to the length of the critical path in the precedence graph, this initial solution is optimal.

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 r 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 $\forall j \in J$ such that $k_{rj} > 0 : \Pi_r \bigcap PPT_j = \emptyset$, the resource $r \in R$ is idle and can be eliminated.

Eliminating non-scarce resources

Schirmer[15] 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:

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

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

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

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

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.[4], 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 [3]. 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.

A filter for variables based on resources

For each activity j and each possible finishing time $t \in E_j$, we compute a lower bound LB_{rjt} on the consumption of each resource r if activity j finishes at time t. We first consider the resource consumption of activity j when finishing at that time t and then we add the minimum consumption of each other activity in the project. For each activity i not linked with j by precedence constraints the minimum is calculated over all the periods in E_i. For an activity k which is a predecessor or successor of j, the set E_k is reduced taking into account that j is finishing at time t. If for some resource r, $LB_{rjt} > K_r$, time t is not feasible for activity j to finish in and the corresponding variable x_{jt} is set to 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 LB_{rjt} on the reduced sets. As the minimum are calculated over restricted subsets, it is possible that new finishing times will fail the test and be eliminated. The process is repeated until no finishing time is eliminated in a complete run.

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 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, let us denote by $\gamma_j = min\{t \,|\, t \in PFT_j\}$. Then, for each i successor of j the finishing times $t \in PFT_i$ such that $t < \gamma_j + d_i$ can also 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. If the makespan of the initial solution built by GRASP equals the minimum finishing time of the last activity n, then this solution is optimal.

4 Scatter Search algorithm

A Scatter Search algorithm is an approximate procedure in which an initial population of feasible solutions is built and then the elements of specific subsets of that population are systematically combined to produce new feasible solutions which will hopefully improve the best known solution (see the book by Laguna and Marti[10] for a comprehensive description of the algorihm). The basic algorithmic scheme is then composed of 5 steps:

1. Generation and improvement of solutions
2. Construction of the Reference Set
3. Subset selection
4. Combination procedure
5. Update of the Reference Set

This basic algorithm stops when the Reference Set cannot be updated and then no new solutions are available for the combination procedure. However, the scheme can be enhanced by adding a new step in which the Reference Set is regenerated and new combinations are possible. The next subsections describe each step of the algorithm in detail.

4.1 Generation and improvement of solutions

The initial population is generated by using a simplified version of the GRASP algorithm developed by Alvarez-Valdes et al.[1]. Here we provide a brief description, while the complete details can be found in [1]. GRASP 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. Therefore, the two phases of GRASP correspond to the generation and improvement of feasible solutions required at the first step of Scatter Search.

The constructive phase

We have adapted the Serial Scheduling Scheme (SSS) proposed by Schirmer[15], 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 randomized priority rule.

Step 0. Initialization

$s = 1$ (stage counter)

$FT_1 = 0$ (sequencing dummy activity 1)

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

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

Step 1. Constructing the set of decisions

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

Step 2. Choosing the decision

Select a decision (j^*, t^*) in D_s, according to a randomized 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 $D_s = \emptyset$, call the Repairing Mechanism

If a feasible decision (j^*, t^*) is found for scheduling $j^* \in EL_s$, go to Step 4

Otherwise, STOP. The algorithm does not find feasible solution

Else, go to Step 2

Step 4. Update

$s = s + 1$

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

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

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

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

If $s = n$, STOP. The schedule 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[15] original scheme. However, we have preferred not to check the resource availability of every decision but only check the decision already chosen. In problems with a large number of possible finishing times for the activities, this strategy is more efficient.

In order to select the priority rule we have tested the 32 priority rules used by Schirmer[15]. 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 led us to choose rule LFT as the most adequate in terms of speed and solution quality. These preliminary results also showed that even with the best performing rules the deterministic constructive algorithm, in which at each step the decision with highest priority value is selected, failed to obtain a feasible solution for many of the instances of 10 activities generated by Böttcher et al.[2]. Therefore, the objective of the randomization procedure 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.

We introduce a randomization procedure for selecting the decision at Step 2. Let s_{jt} be the score of decision (j,t) on the priority rule and $s_{max} = max\{s_{jt}|(j,t) \in D_s\}$, and let δ be a parameter to be determined $(0 < \delta < 1)$. We build a restricted candidate list $S = \{(j,t) \mid s_{jt} \geq \delta s_{max}\}$ and perform a biased selection on 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 activities involved will be equally likely and the randomized selection process will favor them.

The randomization strategy significantly improved the ability of the constructive algorithm to find feasible solutions for tightly constrained instances. However, a limited computational experience showed that the constructive algorithm could still not obtain feasible solutions for all the instances of 10 activities generated by Böttcher et al.[2]. Therefore, we decided to include a repairing mechanism for unfeasible partial schedules. If at Step 3 all decisions in D_n fail the feasibility test and D_n becomes empty, instead of stopping the process and starting a new iteration, we try to re-assign some of the activities already sequenced to other finishing times in order to free some resources that could be used for one of the unscheduled activities to be processed. If this procedure succeeds, the constructive process continues. Otherwise, it stops.

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 to lower finishing times in such a way that the resulting sequence is feasible according to precedence and resource constraints. We have designed two types of moves: a simple move, involving only the critical activity, and a double move in which, apart from the critical activity, other activities are also moved.

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($\exists j \in M \,|\, s_j = 1$) {

   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.

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 there are several possible new finishing times for an activity, that with minimum global resource consumption is chosen.

3. *Double move*

   The activities in M are considered in topological order for moving to the left. For each activity $j \in M$, all possible finishing times earlier than its current finishing time, which satisfy the precedence constraints, are studied. If a new finishing time t is resource-feasible, j is moved to finish at t and no other activity needs to be moved. If this is not the case, the other activities in J 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 the end of J without completely covering the deficit created by moving j, but this deficit is partially reduced, we can go back 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 ($improve = true$)

4.2 Generation of the Reference Set

From the initial population, a set of b solutions is selected to form the Reference Set, S, the set of solutions which will be combined to obtain new solutions. Following the usual strategy, b_1 of them are selected according to a quality criterion: the b_1 solutions with shortest makespans, with ties randomly broken. The remaining $b_2 = b - b_1$ solutions are selected according to a diversity criterion: the solutions are selected one at a time, each one of them the most diverse from the solutions currently in the Reference Set. That is, select solution s^* for which the $Min_{s \in S}\{dist(s, s^*)\}$ is maximum. The distance between two solutions s_1 and s_2 is defined as

$$dist(s_1, s_2) = \sum_{i=1}^{n} |s_1^i - s_2^i|$$

where s_j^i is the finishing time of the $i-th$ activity in solution s_j.

4.3 Subset selection

Several combination procedures were developed and tested. Most of them combine 2 solutions, but one of them combines 3 solutions. The first time the combination procedure is called, all pairs (or trios) of solutions are considered and combined. In

the subsequent calls to the combination procedure, when the Reference Set has been updated and is composed of new and old solutions, only combinations containing at least one new solution are studied.

4.4 Combining solutions

Eight different combination procedures have been developed. Each solution s_j is represented by the vector of the finishing times of the n activities of the project: $s_j = (s_j^1, s_j^2, ..., s_j^n)$. When combining 2 solutions s_1 and s_2 (or 3 solutions s_1, s_2 and s_3), the solutions will be ordered by nondecreasing makespan. Therefore, s_1 will be a solution with makespan lower than or equal to the makespan of s_2 (and the makespan of s_2 will be lower than or equal to the makespan of s_3).

Combination 1

The finishing times of each activity in the new solution, s_c, will be a weighted average of the corresponding finishing times in the two original solutions:

$$s_c^i = \lfloor \frac{k_1 s_1^i + k_2 s_2^i}{k_1 + k_2} \rfloor \quad \text{where } k_1 = (1/s_1^n)^2 \text{ and } k_2 = (1/s_2^n)^2$$

Combination 2

A crosspoint $m \in \{1, 2, .., n\}$ is taken randomly. The new solution, s_c, takes the first m finishing times from s_1. The remaining finishing times $m+1, m+2, .., n$ are selected at random from s_1 or s_2.

Combination 3

A crosspoint $m \in \{1, 2, .., n\}$ is taken randomly. The new solution, s_c, takes the first m finishing times from s_1. The remaining finishing times $m+1, m+2, .., n$ are selected from s_1 or s_2 with probabilities, p_1 and p_2, inversely proportional to the square of their makespan:

$$p_1 = \frac{(1/s_1^n)^2}{(1/s_1^n)^2 + (1/s_2^n)^2} \quad \text{and} \quad p_2 = \frac{(1/s_2^n)^2}{(1/s_1^n)^2 + (1/s_2^n)^2}$$

Combination 4

The combination procedure 2 with $m = 1$. Only the first finishing time is guaranteed to be taken from s_1.

Combination 5

The combination procedure 3 with $m = 1$. Only the first finishing time is guaranteed to be taken from s_1.

Combination 6

Two crosspoints $m_1, m_2 \in \{1, 2, .., n\}$ are taken randomly. The new solution, s_c, takes the first m_1 finishing times from s_1. The finishing times $m_1+1, m_1+2, .., m_2$ are taken from s_2 and the finishing times $m_2+1, m_2+2, .., n$ are taken from s_1.

Combination 7

The finishing times of s_1 and s_2 are taken alternatively to be included in s_c. Starting from the last activity n, $s_c^n = s_1^n$, then $s_c^{n-1} = s_2^{n-1}$ and so on until completing the combined solution.

Combination 8

Three solutions s_1, s_2 and s_3 are combined by using a voting procedure. When deciding the value of s_c^i, the three solutions vote for their own finishing time s_1^i, s_2^i, s_3^i. The value with a majority of votes is taken as s_c^i. If the three values are different, there is a tie. In that case, if the makespan of s_1 is strictly lower than the others, the vote of quality of s_1 imposes its finishing time. Otherwise, if two or three solutions have the same minimum makespan, the finishing time is chosen at random from among those of the tied solutions.

$$s_c^i = \begin{cases} s_1^i & \text{if } s_1^i = s_2^i \\ s_2^i & \text{if } s_1^i \neq s_2^i = s_3^i \\ s_1^i & \text{if } s_1^i \neq s_2^i \neq s_3^i \text{ and } s_1^n < s_2^n \\ random\{s_1^i, s_2^i\} & \text{if } s_1^i \neq s_2^i \neq s_3^i \text{ and } s_1^n = s_2^n < s_3^n \\ random\{s_1^i, s_2^i, s_3^i\} & \text{if } s_1^i \neq s_2^i \neq s_3^i \text{ and } s_1^n = s_2^n = s_3^n \end{cases}$$

Most of the solutions obtained by the combination procedures do not satisfy all the resource and precedence constraints. The non-feasible solutions go through a repairing process that tries to produce feasible solutions as close as possible to the non-feasible combined solution. This process is composed of two phases. First, the finishing times s_c^i are considered in topological order to check if the partial solution $(s_c^1, s_c^2, .., s_c^i)$ satisfies precedence and resource constraints. If that is the case, the next time s_c^{i+1} is studied. Otherwise, s_c^i is discarded as the finishing time of activity i and a new time is searched for from among those possible finishing times of i. The search goes from times close to s_c^i to times far away from it. As soon as a time t^i is found which could be included in a feasible partial solution $(s_c^1, s_c^2, .., t^i)$, the search stops and the next time s_c^{i+1} is considered. If no feasible time is found for activity i, the process goes to the second phase which consists of a repairing procedure similar to that of the constructive algorithm. This procedure tries to change the finishing times of previous activities, $2, 3, .., i-1$, in order to give activity i more chances of finding a finishing time satisfying precedence and resource constraints. If this repairing mechanism succeeds, the process goes back to the first phase and the next time s_c^{i+1} is considered. Otherwise, the combined solution is discarded.

4.5 Updating the Reference Set

The combined solutions which were initially feasible and the feasible solutions obtained by the repairing process described above go through the improvement phase in Section 4.1. The improved solutions are then considered for inclusion in the Reference

Set. The Reference Set S is updated according to the quality criterion: the best b solutions from among those currently in S and from those coming from the improvement phase will form the updated set S.

If the set S is not updated because none of the new solutions qualify, then the algorithm stops, unless the regeneration of S is included in the algorithmic scheme.

4.6 Regenerating the Reference Set

The regeneration of Reference Set S has two objectives. On the one hand, introducing diversity into the set, because the way in which S is updated may cause diverse solutions with high makespans to be quickly substituted by new solutions with lower makespans but more similar to solutions already in S. On the other hand, obtaining high quality solutions, even better than those currently in S.

The new solutions are obtained by again applying the GRASP algorithm described in Section 4.1, with a modification. We take advantage of the information obtained up to that point about the optimal solution in order to focus the search on high quality solutions. More precisely, if the best known solution has a makespan s^n_{best}, we set the planning horizon $T = s^n_{best}$ and run the preprocessing procedures again, reducing the possible finishing times of the activities. When we run the GRASP algorithm, obtaining solutions is harder, because only solutions with makespan lower than or equal to s^n_{best} are allowed, but if the algorithm succeeds we will get high quality solutions.

For the regenerated set S we then consider three sources of solutions: the solutions obtained by the GRASP algorithm, the solutions currently in S and the solutions in the initial population. From these solutions, the new set S is formed as described in Section 4.2. Typically, the b_1 quality solutions will come from the solutions obtained by the GRASP, completed if necessary by the best solutions already in S, while the b_2 diverse solutions will come from the initial population.

5 Computational results

5.1 Test instances

Böttcher et al.[2] generated a first set of test instances. Taking as their starting point PROGEN 2 [9], 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[15] 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.[2] failed to obtain a feasible solution. Table 1 shows the status of Schirmer's problems as reported in [15]

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*

5.2 Preprocessing results

The preprocessing procedures in Section 3 have been applied to the Böttcher et al.[2] problems of 10, 15, 20, 25 and 30, which are available on 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.

	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*

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 from that appearing in Section 2. All the 76 undecided instances of size 10 are impossible, the undecided instance of size 15 has solution and the 2 undecided instances of size 25 are impossible. For the 2 instances of size 20 and the 3 instances of size 30 left undecided by the preprocessing, long time runs of CPLEX 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. In this Table the problems left undecided have not been included, because it is highly unlikely that they have any feasible solution. For more than 70 % of the instances the optimal solutions are found by preprocessing techniques.

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.

	n=10	n=15	n=20	n=25	n=30
Problems	2160	250	250	250	250
Feasible problems	879	234	231	236	239
Solved to optimality by pre-processing	646	165	177	190	193
Remaining problems	233	67	54	46	46

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

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.

	n=10	n=15	n=20	n=25	n=30
Problems	233	67	54	46	46
Initial resources	30	30	30	30	30
Remaining resources (on average)	18 (60%)	18 (60%)	23 (77%)	25 (83%)	25 (83%)
Initial variables (on average)	90	268	565	874	1314
Remaining variables (on average)	51 (57%)	130 (49%)	348 (62%)	611 (70%)	906 (69%)

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

Similar results have been obtained for the test problems generated by Schirmer[15]. Table 5 shows the performance of preprocessing, first determining the status of all of the problems and then providing optimal solutions for many of them. Note that the status of all the problems left undecided in Schirmer's book[15] 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.

The reductions in 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 6 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 6: *Schirmer problems - Reductions of resources and variables*

5.3 Computational results of Scatter Search algorithms

In order to obtain the initial population, the GRASP algorithm is run until 100 different feasible solutions are obtained or the limit of 2000 iterations is reached. From the initial population, a reference set S of $b = 10$ solutions is built, with $b_1 = 5$ quality solutions and $b_2 = 5$ diverse solutions.

The 8 combination procedures described in Section 4.4 have been tested on Schirmer's problems and the results appear in Table 7. For each procedure the number of non-optimal solutions is provided. In this preliminary experience, no regeneration of the reference set is included.

Combination	Non-optimal solutions				
procedure	n=10	n=20	n=30	n=40	Total
1	3	25	39	61	128
2	3	24	42	61	130
3	2	28	43	61	134
4	4	26	43	57	130
5	2	27	44	56	129
6	3	26	42	59	130
7	6	43	66	80	195
8	3	22	41	61	127

Table 7: *Comparison of combination procedures on Schirmer's problems*

Apart from the bad results of procedure 7, Table 7 shows that the other methods obtain similar results. The reason may lie in the fact that many of the solutions resulting from the combination procedures are unfeasible and must be repaired, and the repairing procedures may produce similar results for different combined solutions. For further testing we keep Combinations 1 and 8, which produce the best results and have completely different structures.

Table 7 also shows that the basic Scatter Search algorithm is very efficient, obtaining optimal solutions for most of the 3826 feasible Schirmer test instances. Therefore, an

additional step in which the reference set is regenerated will only be justified if it helps to solve the hardest problems, those not solved by the basic algorithm. The regeneration procedure described in Section 4.6 depends on three parameters: the number of iterations of the modified GRASP algorithm, the number of new solutions obtained, and the number of times the regeneration process is called. We have considered the following values for these parameters:

1. Number of iterations: 500 - 1000
2. Number of solutions: 20 - 50
3. Calls to regenerate: 3 times - Only when the solution is improved after the last call to regenerating

We have tested 6 combinations of these parameters on those Schirmer problems not solved by the the basic algorithm with combination methods 1 or 8. Tables 8 and 9 show the overall results on the remaining 148 problems for Combination procedures 1 and 8 respectively. For each regeneration strategy the Tables present the average and maximum percentage distances to optimal or best known solutions, because the optimal solution is not known for 1 instance of size 30 and for 5 instances of size 40. The last column gives the average running time, in seconds, on a Pentium IV at 2.8 GHz.

Reg	Iter.	Solutions	Regeneration	Non-optimal solutions	Mean dist. to optimum	Max. dist. to optimum	Average time (secs)
0			No regeneration	128	3.19	18.06	35.1
1	500	20	While improving	100	2.17	15.22	66.6
2	1000	20	While improving	94	2.02	15.22	84.5
3	1000	50	While improving	90	1.84	15.22	96.9
4	500	20	3 times	66	1.39	13.04	129.5
5	1000	20	3 times	62	1.30	13.04	172.7
6	1000	50	3 times	60	1.23	15.22	191.3

Table 8: *Comparison of regeneration procedures on hard Schirmer problems (Combination method 1)*

If we compare both Tables, line by line, we can see that the regeneration strategies work better when combined with the Combination procedure 8 in terms of non-optimal solutions and average distance to optimum, though requiring longer computing times due to the combination of three solutions at each step. In Table 9 we can see that the last alternative, Reg 6, obtains very good results, optimally solving more than 50% of the remaining problems. However, it requires long computing times and, as the number of calls to regenerate is fixed, it will regenerate the reference set three times, even for instances for which the basic algorithm had found (though not proved) the optimal solution. Therefore, for the final computational experience, we also keep the alternative Reg 1, in which the regeneration procedure is called only while the solution is improved,

Reg	Iter.	Solutions	Regenerate	Non-optimal solutions	Mean dist. to optimum	Max. dist. to optimum	Average time (secs)
0			No regeneration	127	3.30	18.06	47.3
1	500	20	While improving	80	1.78	13.04	86.0
2	1000	20	While improving	76	1.66	13.04	111.0
3	1000	50	While improving	73	1.55	13.04	119.6
4	500	20	3 times	62	1.32	13.04	158.4
5	1000	20	3 times	58	1.14	13.04	210.6
6	1000	50	3 times	53	1.10	13.04	225.3

Table 9: *Comparison of regeneration procedures on hard Schirmer problems (Combination method 8*

and also restrict the maximum running time to 300 seconds. This time limit, which is checked only at the end of each phase, will produce a slight deterioration in the overall results, but the average times will be greatly reduced by cutting the extremely long running times of some instances.

Tables 10 and 11 show the complete results of the three versions of the Scatter Search algorithm: Reg 0, Reg 1, Reg 6, and compare them with the GRASP algorithm of Alvarez-Valdes et al.[1]. Table 10 contains the results on the 3826 feasible Schirmer instances, while Table 11 presents the results on the 1819 feasible Böttcher et al. problems. The first part of the Tables shows the average distance to optimal solutions (or best known solutions, because the optimum is unknown for 6 Schirmer and 15 Böttcher et al. instances). The second part shows the number of times the best solution does not match the optimal or best known solution, while the third part shows the average computing times in seconds. Both Tables show that, while the GRASP algorithm is very efficient, the Scatter Search procedure with increasingly complex regeneration strategies can significantly improve the results with a moderate increase in the running times.

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 for modelling 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

Problem size	Instances	Scatter Search Algorithm Regen 0	Regen 1	Regen 6	GRASP algorithm
		Average deviation from optimal solution			
10	946	0.02	0.00	0.00	0.01
20	960	0.09	0.03	0.02	0.07
30	960	0.15	0.09	0.05	0.11
40	960	0.24	0.14	0.10	0.23
Total	3826	0.13	0.07	0.04	0.11
		Non-optimal solutions			
10	946	3	0	0	2
20	960	22	10	5	19
30	960	41	29	21	34
40	960	61	41	31	53
Total	3826	127	80	57	108
		Average running time			
10	946	1.1	1.6	2.1	1.0
20	960	1.8	3.4	10.1	0.7
30	960	3.5	5.7	13.8	2.1
40	960	6.6	10.6	20.7	4.4
Total	3826	3.3	5.3	11.7	2.1

Table 10: *Comparison of Scatter Search and GRASP algorithms on Schirmer problems*

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 that point 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, GRUPOS03/174.

Problem size	Instances	Scatter Search Algorithm Regen 0	Regen 1	Regen 6	GRASP algorithm
		Average deviation from optimal solution			
10	879	0.01	0.01	0.00	0.01
15	234	0.14	0.11	0.11	0.09
20	231	0.33	0.29	0.29	0.33
25	236	0.33	0.33	0.21	0.37
30	239	0.10	0.03	0.06	0.16
Total	1819	0.12	0.10	0.09	0.13
		Non-optimal solutions			
10	879	1	1	0	2
15	234	6	4	4	3
20	231	8	7	7	8
25	236	7	7	6	9
30	239	6	4	4	6
Total	1819	28	23	21	28
		Average running time			
10	879	0.5	0.5	0.9	0.2
15	234	1.3	1.6	2.2	1.2
20	231	6.9	7.9	12.7	3.0
25	236	15.9	18.4	24.6	6.8
30	239	16.6	20.0	25.4	9.0
Total	1819	5.5	6.5	8.9	2.7

Table 11: *Comparison of Scatter Search and GRASP algorithms on Böttcher et al. problems*

References

[1] R. Alvarez-Valdes, E. Crespo, J.M. Tamarit, F. Villa, GRASP and Path Relinking for Project Scheduling under Partially Renewable Resources, Technical Report 2004-10, Department of Statistics and Operations Research, University of Valencia, Spain.

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

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

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

[9] 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.

[10] M.Laguna, R.Marti, Scatter Search, Kluwer Academic Publishers, Boston, 2004.

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

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

[13] 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.

[14] 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.

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

[16] 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.

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