Chapter 11
Partially Renewable Resources

Ramon Alvarez-Valdes, Jose Manuel Tamarit, and Fulgencia Villa

Abstract In recent years, in the field of project scheduling 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 into classical models, but can be found in real problems in timetabling and labor scheduling. In this chapter we define this type of resource, describe an integer linear formulation and present some examples of conditions appearing in real problems which can be modeled using partially renewable resources. Then we introduce some preprocessing procedures to identify infeasible instances and to reduce the size of the feasible ones. Some exact, heuristic, and metaheuristic algorithms are also described and tested.

Keywords GRASP • Makespan minimization • Partially renewable resources • Project scheduling • Scatter search

11.1 Introduction

The classical 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. However, in the last decade new types of resources have been proposed to allow the model to include new types of constraints which appear in industrial problems: allocatable resources (Schwindt and Trautmann

R. Alvarez-Valdes (✉) • J.M. Tamarit
Department of Statistics and Operations Research, University of Valencia, Valencia, Spain
e-mail: ramon.alvarez@uv.es; jose.tamarit@uv.es

F. Villa
Department of Applied Statistics and Operations Research, and Quality, Polytechnic University of Valencia, Valencia, Spain
e-mail: mfuvilju@eio.upv.es

C. Schwindt, J. Zimmermann (eds.), *Handbook on Project Management and Scheduling Vol. 1*, International Handbooks on Information Systems,
DOI 10.1007/978-3-319-05443-8_11

2003, Mellentien et al. 2004), storage or cumulative resources (Neumann et al. 2002, 2005, Chap. 9 of this handbook), spatial resources (de Boer 1998), recycling resources (Shewchuk and Chang 1995), time-varying resources (Chap. 8 of this handbook) or continuous resources (Chap. 10 of this handbook).

Another new type of resource is partially renewable resources. The availability of this 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 as 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. The RCPSP with partially renewable resources is denoted by RCPSP/π or in the three-field classification by $PSp|prec|C_{max}$.

Partially renewable resources were first introduced by Böttcher et al. (1999), who proposed an integer linear formulation and developed exact and heuristic algorithms. Schirmer (2000) studied this new type of resource thoroughly in his book on project scheduling problems. He presented many examples of special conditions which can be suitably modeled using partially renewable resources and made a theoretical study of the problem. He also proposed several families of heuristic algorithms for solving the problem. More recently, Alvarez-Valdes et al. (2006, 2008) have developed some preprocessing procedures, as well as GRASP/Path Relinking and Scatter Search algorithms, which efficiently solve the test instances published in previous studies.

In this chapter we introduce the concept of partially renewable resources and review all the existing algorithms. In Sect. 11.2 we define first this type of resource and describe an integer linear formulation, adapted from Böttcher et al. (1999). Then we present and discuss some examples of conditions appearing in real problems which can be modeled using partially renewable resources, and briefly review the first proposed algorithms. Section 11.3 contains the preprocessing procedures and Sects. 11.4 and 11.5 describe the GRASP/PR and the Scatter Search algorithms in more detail. Finally, Sect. 11.6 contains the computational results and Sect. 11.7 the conclusions.

11.2 Solving Problems with Partially Renewable Resources

The RCPSP/π can be defined as follows: Let n be the number of activities to schedule. The project consists of $n+2$ activities, numbered from 0 to $n+1$, where activities 0 and $n+1$ represent the beginning and the end of the project, respectively.

Let $Pred(i)$ be the set of immediate predecessors of activity i. Each activity i has a processing time of p_i and once started cannot be interrupted. Let $\mathscr{R}^p$ be the set of partially renewable resources. Each resource $k \in \mathscr{R}^p$ has a total availability R_k and an associated set of periods Π_k. An activity i requiring resource k will consume r_{ik} units of it at each period $t \in \Pi_k$ in which it is processed. Finally, let T be the planning horizon in which all the activities must be processed. For each activity i we obtain its earliest and latest start times, ES_i, LS_i, by critical path analysis. We denote by $W_i = \{ES_i, \dots, LS_i\}$, the set of possible starting times, and $Q_{it} = \{t - p_i + 1, \dots, t\}$.

11.2.1 Formulation of the Problem

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:

$$x_{it} = \begin{cases} 1 & \text{if activity } i \text{ starts at time } t \\ 0 & \text{otherwise.} \end{cases}$$

the problem can be formulated as follows:

$$\text{Min.} \sum_{t \in W_{n+1}} t\, x_{n+1,t} \tag{11.1}$$

$$\text{s. t.} \sum_{t \in W_i} x_{it} = 1 \qquad (i = 0, \dots, n+1) \tag{11.2}$$

$$\sum_{t \in W_j} (t + p_j) x_{jt} \le \sum_{t \in W_i} t\, x_{it} \qquad (i = 0, \dots, n+1, j \in Pred(i)) \tag{11.3}$$

$$\sum_{i=1}^{n} r_{ik} \sum_{t \in \Pi_k} \sum_{q \in Q_{it} \bigcap W_i} x_{iq} \le R_k \qquad (k \in \mathscr{R}^p) \tag{11.4}$$

$$x_{it} \in \{0, 1\} \qquad (i = 0, \dots, n+1, \ \ t \in W_i) \tag{11.5}$$

The objective function (11.1) to be minimized is the starting time of the last activity and hence the makespan of the project. According to constraints (11.2), each activity must start once. Constraints (11.3) are the precedence constraints and constraints (11.4) are the resource constraints. Note that in this problem there is only one global constraint for each resource $k \in \mathscr{R}^p$. An activity i consumes r_{ik} units of a resource k for each period in which it is processed within Π_k. Another special characteristic of this problem is that all the activities must finish within the planning horizon T in which sets Π_k are defined. The starting times of each activity i form a finite set W_i and therefore the existence of feasible solutions is not guaranteed. In fact, Schirmer (2000) has shown that the feasibility variant of the RCPSP/π is $\mathcal{NP}$-complete in the strong sense.

The above formulation is called the normalized formulation by Böttcher et al. (1999) and Schirmer (2000). Both consider an alternative, more general, formulation in which several subsets of periods are associated with each resource $k \in \mathcal{R}^p$. $\Pi_k = \{P_{k1}, \ldots, P_{k\nu}\}$ is a set of subsets of periods where each subset of periods $P_{k\mu} \in \Pi_k$ has a resource capacity $R_{k\mu}$. This general formulation can be transformed into the above formulation by a "normalization" procedure in which a new resource k' is defined for each subset of periods $P_{k\mu}$. Therefore, the normalized formulation can be used without loss of generalization.

11.2.2 *Modeling Capabilities*

In this section we describe several situations in which partially renewable resources can be used to model conditions which classical renewable or non-renewable cannot accommodate.

11.2.2.1 Lunch Break Assignments

In this example, taken from Böttcher et al. (1999), we have five workers in a 09:00 to 18:00 shift with a lunch break of 1 h. If the lunch period is fixed to period 13 for all workers, the availability profile would be that depicted in Fig. 11.1. We would get more flexibility if each worker could have the break either in period 13 or 14. However, if we use renewable resources, the number of workers taking the break at each period must have been decided beforehand. An example appears in Fig. 11.2, in which three workers take the break at period 13 and two at period 14.

The situation can be more appropriately modeled by using a partially renewable resource k with $\Pi_k = \{13, 14\}$ and total availability of $R_k = 5$ units. Each task consumes $r_{ik} = 1$. Therefore, the total number of working hours assigned to these periods cannot exceed five, leaving time for the five workers to have their lunch break.

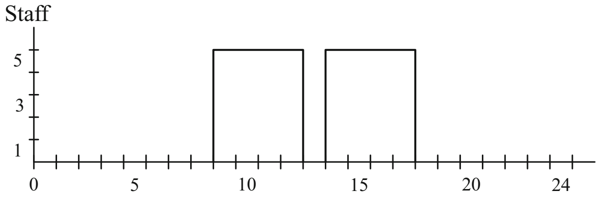

Fig. 11.1 Fixed break at period 13

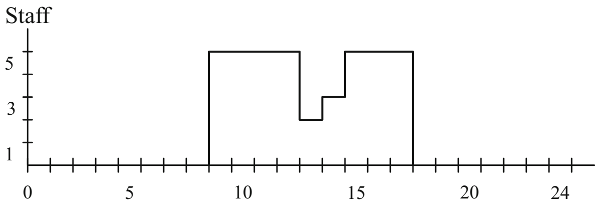

Fig. 11.2 Fixed distribution of breaks between periods 13 and 14

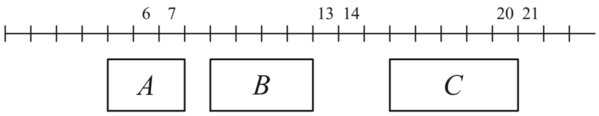

Fig. 11.3 Weekend days-off assignment

11.2.2.2 Weekend Days-Off Assignments

Let us consider a contractual condition like that of *working at most two weekend days out of every three consecutive weeks* (Alvarez-Valdes et al. 2008). This condition cannot be modeled as a renewable resource because this type of resource considers each period separately. It cannot be modeled as a non-renewable resource because this type of resource considers the whole planning horizon. We model this condition for each worker as a partially renewable resource k with a set of periods $\Pi_k = \{6, 7, 13, 14, 20, 21\}$ for the first three weekends and a total availability of $R_k = 2$ units. Each task i assigned to this worker consumes $r_{ik} = 1$ unit of this resource for each weekend day in which it is processed. In Fig. 11.3 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 two units of the resource. Activity B does not consume the resource and activity C consumes one unit in period 20. If these three activities had to be done by the same worker, the solution in the figure would not be possible because it would exceed the resource availability.

11.2.2.3 Special Conditions in School Timetabling Problems

When building their timetables, some schools include conditions such as that each teacher must have classes at least one afternoon per week and at most two afternoons per week. Let us assume for simplicity that the morning sessions are numbered 1, 3, 5, 7, 9 and the afternoon sessions are numbered 2, 4, 6, 8, 10. For each teacher we define two partially renewable resources. In both cases, $\Pi_1 = \Pi_2 = \{2, 4, 6, 8, 10\}$. In order to ensure that the teacher attends at least one afternoon session, $R_1 = -1$ and for each of the teacher's classes, $r_{i1} = -1$. The second condition is ensured defining $R_2 = 2, r_{i2} = 1$.

A complete study of logical conditions that can be modeled by using partially renewable resources and then applied to different real situations can be found in Schirmer (2000).

11.2.3 A Branch & Bound Algorithm

Böttcher et al. (1999) proposed an exact algorithm for the RCPSP/π that is based upon the branch and bound method introduced by Talbot and Patterson (1978) for the classical RCPSP, because this was one of the few approaches allowing time-varying capacity profiles to be accommodated. The basic scheme of the algorithm is straightforward. Starting with an empty schedule, partial schedules are feasibly augmented by activities, one at a time. The enumeration tree is built with a depth-first strategy, where the depth of a branch is the number of activities scheduled. Backtracking occurs when an activity cannot be scheduled within its feasible interval. In order to reduce the computational effort, Böttcher et al. (1999) developed several resource-based feasibility bounds tailored to the specifics of the problem.

The authors employed a sample of 1,000 test instances generated from a modification of ProGen (Kolisch et al. 1995). They generated four sets with 250 instances each one and with 15, 20, 30, and 60 activities, respectively. The number of resources for each set was 30. Applying the branch and bound algorithm in a truncated version (TBB), using a CPU time limit of 5 min per instance on a C-language implementation on an IBM RS/6000 workstation at 66 MHz, produced the results shown in Table 11.1.

The study by Böttcher et al. (1999) also included several priority rules designed to obtain good quality solutions in short computing times. First, they tested some classical rules: MINEFT (minimum earliest finishing time), MINLFT (minimum latest finishing time), MINSLK (minimum slack), MTSUCC (most total successors). Defining the maximum resource usage of each activity i as $SRU_i = \sum_{k \in \mathscr{R}^p} r_{ik}$, they included the priority rules MAXSRU and MINSRU, considering the maximum and minimum of the activities resource consumptions. In order to obtain a more accurate calculation of resource usage they defined SC_{ikt}, the consumption of resource k by activity i when started at period t, $MC_{ikt} = \min\{SC_{ik\tau} | t \leq \tau \leq LS_i\}$, the minimum resource consumption of resource k by activity i when started not earlier than t and not later than LS_i, and MCI_{ikt}

Table 11.1 Effectiveness of TBB algorithm proposed by Böttcher et al. (1999)

Activities	Solved to optimality	Solved to feasibility	Unsolved
15	240	10	–
20	229	21	–
30	217	14	19
60	229	11	10

which is the sum of MC_{ikt} of all the successors of activity i considering their corresponding time intervals. These elements were used to calculate $TRU_{it} = \sum_{k \in \mathscr{R}^p} (SC_{ikt} + MCI_{ikt})$ and $RRU_{it} = \sum_{k \in \mathscr{R}^p : R_k > 0} (SC_{ik} + MCI_{ikt})/R_k$ which are lower bounds on the absolute and relative resource usage, producing rules MAXTRU, MINTRU, MAXRRU, MINRRU. These rules are static because they do no take into account the resources already consumed in a partial solution. If R_k^0 if the left-over capacity of resource k with respect to a partial schedule, $DRRU_{jt} = \sum_{k \in \mathscr{R}^p : R_k^0 > 0} (SC_{ikt} + MCI_{ikt})/R_k^0$ is the dynamic relative resource usage and $TRC_{jt} = \sum_{k \in \mathscr{R}^p} (R_k^0 - SC_{ikt} + MCI_{ikt})/R_k^0$ an upper bound of the total remaining capacity. These values were used to define rules MAXDRRU, MINDRRU, MAXTRC, MINTRC. Böttcher et al. (1999) concluded that classical rules worked well for easy instances, while the new dynamic rules did a better job on hard instances. In summary, rule MINLFT seemed to be the most promising one.

11.2.4 Schirmer's Algorithms

In his book in 2000, Schirmer made a complete study of partially renewable resources, their modeling capabilities and theoretical properties. He also developed a series of progressively more complex algorithms. First, he collected known priority rules and added some new ones. The list appears in Table 11.2. In order to test them, Schirmer used Progen to generate new instances, but previously he analyzed the parameter combinations that generally produce infeasible solutions. So he selected the most promising parameter combinations to generate the new set of test instances. For each of the 96 promising clusters, ten instances were generated of sizes 10, 20, 30, and 40 with 30 resources.

In Table 11.2, RK_{ks} is the remaining availability or resource k at stage s; RD_{ikt} is the relevant demand (consumption) of resource k if activity i starts at time t; MDE_{ikt} is the minimum relevant demand on resource k of all the successors of activity i when it starts at time t; and D_s is the set of possible assignments (i, t) at stage s. The 12 rules involving resources can use all the resources or only those which are potentially scarce. Therefore, a total of 32 rules were defined.

In a second phase he introduced randomization, using the Modified Regret-Based Biased Random Sampling, in which the probability assigned to each activity is calculated using the regret, that is the worst possible consequences that might result from selecting another candidate. In a third phase, Schirmer developed a local search, based on local and global left-shifts of the activities. Finally, he proposed a Tabu Search algorithm. He compared the results with those obtained with the *TBB* algorithm by Böttcher et al. (1999) and concluded that good construction methods solved significantly more instances optimally than tabu search, so these perform better on easier instances. Yet, on very hard instances, tabu search outperformed all construction methods. Indeed, the multi-start version of the tabu search method was the only one that managed to solve all the instances attempted, except for five that he conjectured to be infeasible.

Table 11.2 Priority rules proposed by Schirmer (2000)

	Measure	Definition	Static vs. dynamic	Local vs. global
MIN	EF_i	$ES_i + p_i$	S	L
MIN	ES_i	ES_i	S	G
MIN	LF_i	$LS_i + p_i$	S	L
MIN	LS_i	LS_i	S	G
MAX	MTS_i	$\|\{i' \| i < i'\}\|$	S	L
MIN	SLK_i	$LS_i - EF_i$	D	L
MIN	SPT_i	p_i	S	L
MIN	$SSLK_i$	$LS_i - ES_i$	S	L
MAX	DRC_{it}	$\sum_k (RK_{ks} - RD_{ikt})$	D	L
MAX	DRC/E_{it}	$\sum_k (RK_{ks} - RD_{ikt} - MDE_{ikt})$	D	G
MIN	DRD_{it}	$\sum_k RD_{ikt}/max\{RD_{i'kt'} \| (i',t') \in D_s$	D	G
MIN	DRD/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/max\{RD_{i'kt'}$ $+MDE_{i'kt'} \| (i',t') \in D_s)$	D	G
MIN	DRS_{it}	$\sum_k RD_{ikt}/RK_{ks}$	D	L
MIN	DRS/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/RK_{ks}$	D	G
MIN	RRD_{it}	$\sum_k RD_{ikt}/(r_{ik} p_i)$	S	L
MIN	RRD/E_{it}	$\sum_k RD_{ikt}/(r_{ik} p_i) + \sum_{i<j} MDE_{jkEST_j}/(r_{jk} p_k)$	D	G
MIN	TRD_{it}	$\sum_k RD_{ikt}$	S	L
MIN	TRD/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})$	S	G
MIN	TRS_{it}	$\sum_k RD_{ikt}/R_k$	S	L
MIN	TRS/E_{it}	$\sum_k (RD_{ikt} + MDE_{ikt})/R_k$	S	G

11.3 Preprocessing

Preprocessing has two objectives. First, helping to decide whether a given instance is infeasible or if it has feasible solutions. If the latter is the case, a second objective is to reduce the number of possible starting 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 infeasible problems and will concentrate their efforts on the relevant elements of the problem.

The preprocessing we have developed includes several procedures:

1. *Identifying trivial problems*
   If the solution in which the starting time of each activity i is set to ES_i is resource-feasible, then it is optimal.
2. *Reducing the planning horizon T*
   For each instance, we are given a planning horizon $(0, T)$. This value plays an important role in the time-indexed problem formulation. In fact, late starting times of the activities, LS_i 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 a 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.

3. *Eliminating idle resources*

   Each resource $k \in \mathscr{R}^p$ is consumed only if the activities requiring it are processed in periods $t \in \Pi_k$. 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_k. In this case, the resource is idle and can be eliminated. More precisely, if we denote the points in time where activity i can be in progress by $\overline{W_i} = \{ES_i, \ldots, LS_i + p_i - 1\}$, and $\forall i \mid r_{ik} > 0 : \Pi_k \bigcap \overline{W_i} = \emptyset$, the resource $k \in \mathscr{R}^p$ is idle and can be eliminated.

4. *Eliminating non-scarce resources*

   Schirmer (2000) distinguishes between scarce and non-scarce resources. He considers a resource $k \in \mathscr{R}^p$ as scarce if $\sum_{i=1}^{n} r_{ik} p_i > R_k$, that is, if an upper bound on the maximum resource consumption exceeds the resource availability. In this case, the upper bound is computed by supposing that all the activities requiring the resource are processed completely inside Π_k.

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

$$\text{Max.} \sum_{i=1}^{n} r_{ik} \sum_{t \in \Pi_k} \sum_{q \in Q_{it} \bigcap W_i} x_{iq} \tag{11.6}$$

$$\text{s.t.} \sum_{t \in W_i} x_{it} = 1 \qquad (i = 1, \ldots, n) \tag{11.7}$$

$$\sum_{\tau=t}^{T} x_{j\tau} + \sum_{\tau=1}^{t+p_j-1} x_{i\tau} \leq 1 \qquad (i = 1, \ldots, n; j \in Pred(i); t \leq T) \tag{11.8}$$

$$x_{it} \geq 0 \qquad (i = 1, \ldots, n; t \in W_i) \tag{11.9}$$

The objective function (11.6) maximizes the resource consumption over the whole project. Constraints (11.7) ensure that each activity starts once. Constraints (11.8) are the precedence constraints. We use this expression, introduced by Christofides et al. (1987), because it is more efficient than the usual precedence constraint. In fact, the reformulation of these constraints produces a linear problem whose coefficient matrix is totally unimodular and thus all the vertices of the feasible region are integer-valued (Chaudhuri et al. 1994). 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.

5. *A filter for variables based on resources*
   For each activity i and each possible starting time $t \in W_i$, we compute a lower bound LB_{kit} on the consumption of each resource k if activity i starts at time t. We first include the resource consumption of activity i when starting at that time t and then for each other activity in the project we consider all its possible starting times, determine for which of them the resource consumption is minimum and add that amount to LB_{kit}. Note that this minimum is calculated over all the periods in W_j for each activity j not linked with i by precedence constraints, but for an activity h which is a predecessor or successor of i, the set W_h is reduced by taking into account that i is starting at time t. If for some resource k, $LB_{kit} > R_k$, time t is not feasible for activity i to start in, and the corresponding variable x_{it} is set to 0.
   When this filter is applied to an activity i, some of its possible starting times can be eliminated. From then on, the set of possible starting times is no longer W_i. We denote by $\underline{W_i}$ the set of starting times passing the filter.
   This filter is applied iteratively. After a first run on every activity and every starting time, if some of the variables are eliminated the process starts again, but this time computing LB_{kit} on the reduced sets. As the minimum resource consumptions are calculated over restricted subsets, it is possible that new starting times will fail the test and can be eliminated. The process is repeated until no starting time is eliminated in a complete run.
6. *Consistency test for finishing times*
   When the above filter eliminates a starting time of an activity i, it is possible that some of the starting times of its predecessors and successors are no longer feasible. For an activity i, $\tau_i = \max\{t \mid t \in \underline{W_i}\}$. Then, for each $j \in Pred(i)$ the starting times $t \in \underline{W_j}$ such that $t > \tau_i - p_j$ can be eliminated. Analogously, $\gamma_i = \min\{t \mid t \in \underline{W_i}\}$. Then, for each $j \in Succ(i)$, the starting times $t \in \underline{W_j}$ such that $t < \gamma_i + p_i$ can also be eliminated.
   This test is also applied iteratively until no more starting times are eliminated. If, after applying these two procedures for reducing variables, an activity i has $\underline{W_i} = \emptyset$, the problem is infeasible. If the makespan of the initial solution built by GRASP equals the minimum starting time of the last activity $n + 1$, then this solution is optimal.
7. *A linear programming bound for the starting time of activity* $n + 1$
   We solve the linear relaxation of the integer formulation of the problem given in Sect. 11.2.1, using only variables and resources not eliminated in previous preprocessing procedures and replacing expression (11.3) with expression (11.8) for the precedence constraints. The optimal value of the linear program, opl_1, is a lower bound for the optimal value of the starting time of activity $n + 1$. Therefore we can set $x_{n+1,t} := 0$ for all $t < \lceil opl_1 \rceil$. If that eliminates some variables with non-zero fractional values, we solve the modified problem and

obtain a new solution opl_2, strictly greater than opl_1, which can in turn be used to eliminate new variables, and the process goes on until no more variables are removed. If the solution value obtained by GRASP equals the updated minimum starting time for activity $n+1$, this solution is optimal. Otherwise, the lower bound can be used to check the optimality of improved solutions obtained in the GRASP process.

11.4 GRASP Algorithm

In this section we describe the elements of our GRASP implementation in detail. The first two subsections contain the constructive randomized phase and the improvement phase. The last two subsections describe an enhanced GRASP procedure and a Path Relinking algorithm operating over the best GRASP solutions obtained. A comprehensive review of GRASP can be found in Resende and Ribeiro (2003).

11.4.1 The Constructive Phase

11.4.1.1 A Deterministic Constructive Algorithm

We have adapted the serial schedule-generation scheme (SSS) proposed by Schirmer (2000), which in turn is an adaptation of the serial schedule-generation scheme commonly used for the classical RCPSP. We denote by S_i the starting time assigned to activity i. At each stage of the iterative procedure an activity is scheduled by choosing from among the current set of decisions, defined as the pairs (i,t) of an activity i and a possible starting time $t \in \underline{W_i}$. The selection is based on a priority rule.

```
Step 0.   Initialization
    $S_0 := 0$ (sequencing dummy activity 0)
    $\mathscr{C} := \{0\}$ (activities already scheduled)
    $\forall k \in \mathscr{R}^p\ :\ RK_k := R_k$ (remaining capacity of resource $k$)
                    $TD_k := \sum_{i=1}^{n} r_{ik} p_i$ (maximum possible demand for $k$)
    $SR := \{k \in \mathscr{R}^p \mid TD_k > RK_k\}$ (set of possible scarce resources)
    $EL :=$ set of eligible activities, i.e. those activities for which activity 0 is the only
    predecessor
Step 1.   Constructing the set of decisions
    $\mathscr{D} := \{(i,t) \mid i \in EL,\ t \in \underline{W_i}\}$
Step 2.   Choosing the decision
    Select the best decision $(i^*, t^*)$ in $\mathscr{D}$, according to a priority rule
```

```
Step 3.  Feasibility test
    if $(i^*, t^*)$ is resource-feasible, go to Step 4.
    else
        $\mathscr{D} := \mathscr{D} \setminus \{(i^*, t^*)\}$
        if $\mathscr{D} = \emptyset$, STOP. The algorithm does not find a feasible solution.
        else, go to Step 2.
Step 4.  Update
    $S_{i^*} := t^*$
    $\mathscr{C} := \mathscr{C} \cup \{i^*\}$
    $EL := (EL \setminus \{i^*\}) \cup \{i \mid Pred(i) \subseteq S\}$
    $\forall h \mid i \in Pred(h) \ : \underline{W_h} = \underline{W_h} \setminus \{\tau \mid t^* + p_{i^*} > \tau\}$
    $\forall k \in \mathscr{R}^p \ : \quad RK_k := RK_k - RD_{ikt}$
                  $TD_k := TD_k - r_{i^*k} p_{i^*}$
                  if $TD_k \leq RK_k$ , then $SR = SR \setminus \{k\}$
    if $|S| = n + 2$, STOP. The sequence is completed.
    else, go to Step 1.
```

At Steps 1 and 2, we follow Schirmer's design, working with the set of decisions $\mathscr{D}$ and choosing from it both the activity to sequence and the time at which it starts. An alternative could have been to first select the activity and then the time, choosing, for instance, the earliest resource-feasible time. However, this strategy has serious drawbacks. First, it is less flexible. Priority rules which take into account the resource consumption could not be used, because this consumption varies depending on the periods in which the activity is processed. Second, as Schirmer has shown in his book, scheduling each activity at its earliest resource-feasible time may not only fail to produce an optimal solution but may also fail to produce feasible solutions. As he says: "delaying activities from their earliest feasible start time is crucial to finding good—even feasible—solutions for some instances".

At Step 3, we perform a complex feasibility test which does not involve only activity i^* but also the unscheduled activities. Following a process which is quite similar to the filter described in Sect. 11.3, we try to assess the effect of scheduling i^* at time t^* by computing an estimation of the global resource consumptions, also considering the minimum consumption of the activities not yet scheduled. If this estimation exceeds the remaining resource availability, t^* is labeled as non-feasible for i^*. This test allows us to avoid decisions which will inevitably produce infeasible solutions in the later stages of the constructive process and our computational experience has told us that significantly improves the proportion of feasible solutions we obtain. However, it has a larger computational cost. Therefore, we do not perform the feasibility test for each decision of $\mathscr{D}$ at Step 1, as in Schirmer's algorithm, but only for the decision chosen at Step 2. In problems with a large number of possible finishing times for the activities, this strategy is more efficient. If this decision fails the feasibility test of Step 3, the second best decision is tested and so on. Few tests are usually required and therefore it is more convenient to take the feasibility test out of Step 1.

The resource availabilities are updated, subtracting RD_{ikt}, the consumption of activity i when starting at period t from the remaining capacity R_k. We also keep the set of possible scarce resources SR updated because some priority rules based on resource consumption only take this type of resource into account.

11.4.1.2 Priority Rules

We have tested the 32 priority rules used by Schirmer (2000). The first eight 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. Twelve of them use all the resources and the other 12 only the scarce resources. A preliminary computational experience 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 of the ten-activity instances generated by Böttcher et al. (1999). Therefore, the objective of the randomization procedures included in the algorithm was not only to produce diverse solutions but also to ensure that for most of the problems the algorithm would obtain a feasible solution.

11.4.1.3 Randomization Strategies

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

1. *Random selection on the Restricted Candidate List, RCL*
   Select decision (i^*, t^*) at random in set $RCL = \{(i,t) \mid sc_{it} \geq \delta sc_{max}\}$
2. *Biased selection on the Restricted Candidate List, RCL*
   We build the Restricted Candidate List as in alternative 1, but instead of choosing at random from among its elements, the decisions involving the same activity i are given a weight which is inversely proportional to the order of their finishing times. For instance, if in RCL 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 RCL corresponding to the lowest starting times of the activities involved will be equally likely and the randomized selection process will favor them.
3. *Biased selection on the set of decisions $\mathscr{D}$*
   We have also implemented the Modified Regret-Based Biased Random Sampling (MRBRS/δ) proposed by Schirmer (2000), in which the decision (i,t) is chosen

from among the whole set $\mathscr{D}$ 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.

11.4.1.4 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 of the ten-activity instances generated by Böttcher et al. (1999). Therefore, we felt that the algorithm was not well-prepared for solving larger problems and we decided to include a repairing mechanism for infeasible partial schedules.

In the construction process, if at Step 3 all the decisions in $\mathscr{D}$ fail the feasibility test and $\mathscr{D}$ finally 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.

11.4.2 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 starting times must be reduced in order to have a new solution with the shortest makespan. These activities are labeled 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.

11.4.3 An Aggressive Procedure

The standard version of our heuristic algorithm starts by applying the preprocessing procedure in Sect. 11.3. 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 whether the best known solution has been 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 to the reduced problem. Obtaining feasible solutions is now harder, but if the procedure succeeds we will get high quality solutions.

The stopping criterion combines two aspects: the number of iterations since the last improvement and the number of iterations since the last call to preprocessing. For the first aspect a low limit of 50 iterations is set. We do not call preprocessing every time the solution is improved, but we do not wait too long to take advantage of the improvements. For the second aspect we set a higher limit of 500 iterations to give the process enough possibilities of improving the best current solution and, at the same time, do not allow it to run for too long a time.

11.4.4 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*. We progressively impose the attributes of the guiding solution onto the intermediate solutions in the path, 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 the initiating and the guiding solutions.

We keep the ten best solutions obtained in the GRASP procedure. We consider each 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 jth 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 become progressively more similar to the guiding solution and more different from the initiating one. In many cases these intermediate solutions will not be feasible. If this is the case, a repairing mechanism similar to that described in Sect. 11.4 is applied. We proceed from activity 1 to activity n, checking for each activity j whether 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.

11.5 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 (2004) for a comprehensive description of the algorithm). The basic algorithmic scheme is composed of five 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 following enumeration describes each step of the algorithm in detail.

1. *Generation and improvement of solutions*
   The initial population is generated by using the basic version of the GRASP algorithm.
2. *Generation of the Reference Set*
   The Reference Set, *RefSet*, the set of solutions which will be combined to obtain new solutions, is built by choosing a fixed number b of solutions from the initial population. Following the usual strategy, b_1 of them are selected according to a quality criterion: the b_1 solutions with the shortest makespan, 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 *RefSet*. That is, select solution S' for which the $\text{Min}_{S \in RefSet}\{dist(S, S')\}$ is maximum. The distance between two solutions S^1 and S^2 is defined as

$$dist(S^1, S^2) = \sum_{i=1}^{n+1} |S_i^1 - S_i^2|$$

   where S_i^{μ} is the starting time of the i-th activity in solution S^{μ}.
3. *Subset selection*
   Several combination procedures were developed and tested. Most of them combine two solutions, but one of them combines three 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. *Combining solutions*

Eight different combination procedures have been developed. Each solution S^{μ} is represented by the vector of the starting times of the activities of the project: $S^{\mu} = (S_0^{\mu}, S_1^{\mu}, S^{\mu_2}, \dots, S_n^{\mu}, S_{n+1}^{\mu})$. When combining two solutions S^1 and S^2 (or three solutions S^1, S^2 and S^3), the solutions will be ordered by nondecreasing makespan. Therefore, S^1 will be a solution with a 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 starting times of each activity in the new solution, S^{new}, will be a weighted average of the corresponding starting times in the two original solutions:

$$S_i^{new} = \lfloor \frac{k_1 S_i^1 + k_2 S_i^2}{k_1 + k_2} \rfloor \quad \text{where } k_1 = (1/S_{n+1}^1)^2 \text{ and } k_2 = (1/S_{n+1}^2)^2$$

Combination 2

A crosspoint $m \in \{1, 2, .., n\}$ is taken randomly. The new solution, S^{new}, takes the first m starting times from S^1. The remaining starting times $m+1, m+2, .., n+1$ 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^{new}, takes the first m starting times from S^1. The remaining starting times $m+1, m+2, .., n+1$ are selected from S^1 or S^2 with probabilities, π_1 and π_2, inversely proportional to the square of their makespan:

$$\pi_1 = \frac{(1/S_{n+1}^1)^2}{(1/S_{n+1}^1)^2 + (1/S_{n+1}^2)^2} \quad \text{and} \quad \pi_2 = \frac{(1/S_{n+1}^2)^2}{(1/S_{n+1}^1)^2 + (1/S_{n+1}^2)^2}$$

Combination 4

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

Combination 5

The combination procedure 3 with $m = 1$. Only the first starting 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^{new}, takes the first m_1 starting times from S^1. The starting times $m_1 + 1, m_1 + 2, .., m_2$ are taken from S^2 and the starting times $m_2 + 1, m_2 + 2, .., n + 1$ are taken from S^1.

Combination 7
The starting times of S^1 and S^2 are taken alternatively to be included in S^{new}. Starting from the last activity $n + 1$, $S^{new}_{n+1} = S^1_{n+1}$, then $S^{new}_n = S^2_n$ 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^{new}_i, the three solutions vote for their own starting time S^1_i, S^2_i, S^3_i. The value with a majority of votes is taken as S^{new}_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 starting time is chosen at random from among those of the tied solutions.

$$S^{new}_i = \begin{cases} S^1_i & \text{if } S^1_i = S^2_i \\ S^2_i & \text{if } S^1_i \neq S^2_i = S^3_i \\ S^1_i & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} < S^2_{n+1} \\ random\{S^1_i, S^2_i\} & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} = S^2_{n+1} < S^3_{n+1} \\ random\{S^1_i, S^2_i, S^3_i\} & \text{if } S^1_i \neq S^2_i \neq S^3_i \text{ and } S^1_{n+1} = S^2_{n+1} = S^3_{n+1} \end{cases}$$

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 which are as close as possible to the non-feasible combined solution. This process is composed of two phases. First, the starting times S^{new}_i are considered in topological order to check if the partial solution $(S^{new}_0, S^{new}_1, .., S^{new}_i)$ satisfies precedence and resource constraints. If that is the case, the next time S^{new}_{i+1} is studied. Otherwise, S^{new}_i is discarded as the starting time of activity i and a new time is searched for from among those possible starting times of i. The search goes from times close to S^{new}_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 $(S^{new}_0, S^{new}_1, .., t^i)$, the search stops and the next time S^{new}_{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 starting times of previous activities, $1, 2, .., i - 1$, in order to give activity i more chances of finding a starting time satisfying precedence and resource constraints. If this repairing mechanism succeeds, the process goes back to the first phase and the next time S^{new}_{i+1} is considered. Otherwise, the combined solution is discarded.

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 Sect. 11.4.2. The improved solutions are then considered for inclusion in the Reference Set. The Reference Set *RefSet* is updated according to the quality

criterion: the best b solutions from among those currently in *RefSet* and from those coming from the improvement phase will form the updated set *RefSet*.
If the set *RefSet* is not updated because none of the new solutions qualify, then the algorithm stops, unless the regeneration of *RefSet* is included in the algorithmic scheme.

6. *Regenerating the Reference Set*
The regeneration of Reference Set *RefSet* has two objectives: on the one hand, introducing diversity into the set, because the way in which *RefSet* is updated may cause diverse solutions with a high makespan to be quickly substituted with new solutions with a lower makespan but more similar to solutions already in *RefSet*; on the other hand, obtaining high quality solutions, even better than those currently in *RefSet*.
The new solutions are obtained by again applying the GRASP algorithm with a modification. We take advantage of the information about the optimal solution obtained up to that point in order to focus the search on high quality solutions. More precisely, if the best known solution has a makespan S_{n+1}^{best}, we set the planning horizon $T = S_{n+1}^{best}$ and run the preprocessing procedures again, reducing the possible starting times of the activities. When we run the GRASP algorithm, obtaining solutions is harder, because only solutions with a makespan lower than or equal to S_{n+1}^{best} are allowed, but if the algorithm succeeds we will get high quality solutions.
For the regenerated set *RefSet* we then consider three sources of solutions: the solutions obtained by the GRASP algorithm, the solutions currently in *RefSet* and the solutions in the initial population. From these solutions, the new set *RefSet* is formed. Typically, the b_1 quality solutions will come from the solutions obtained by the GRASP, completed if necessary by the best solutions already in *RefSet*, while the b_2 diverse solutions will come from the initial population.

11.6 Computational Results

This section describes the test instances used and summarizes the results obtained on them by the preprocessing procedures, the priority rules, and the metaheuristic algorithms developed in previous sections.

11.6.1 Test Instances

Böttcher et al. (1999) generated an instance generator PROGEN 2. Taking as their starting point PROGEN (Kolisch et al. 1995), an instance generator for the classical RCPSP with renewable resources, they modified and enlarged the set of parameters and generated a set of 2,160 instances with ten non-dummy activities, ten replications for each one of the 216 combinations of parameter values. As most of the problems were infeasible, they restricted the parameter values to the 25 most promising combinations and generated 250 instances of sizes 15, 20, 30 and 60 of non-dummy activities, always keeping the number of resources to 30.

Later Schirmer (2000) developed PROGEN 3, an extension of PROGEN 2, and generated some new test instances. He generated 960 instances of sizes 10, 20, 30, and 40, with 30 resources. Most of them have a feasible solution, while a few of those with ten activities are infeasible. Additionally, we generated a new set of test instances of 60 activities using Schirmer's PROGEN 3, with the same parameter values he used to generate his problems. This new set gave us an indication of the performance and running times of our algorithms on larger problems.

We applied all the algorithms and procedures described in previous sections to both sets of instances, obtaining similar results. In the next subsections we present and comment on the results obtained on Schirmer's problems.

11.6.2 Preprocessing Results

Table 11.3 shows the detailed results of the preprocessing process. A first filter identifies the trivial instances. For the remaining instances, a first feasible solution is obtained. Let us suppose that the value of this solution is S_{n+1}^{start}. If S_{n+1}^{start} equals the length of the critical path, the solution is optimal. The number of instances proven optimal by this test appears as *CPM_bound*. Otherwise, the filters for resources and variables are used, sometimes eliminating some feasible finishing times for the last activity $n + 1$. If S_{n+1}^{start} equals the minimum possible finishing time of $n + 1$, the solution is optimal. The number of instances proven optimal by this test appears as *MIN_PFT_bound*. Finally, the linear bound is calculated. If S_{n+1}^{start} equals the linear bound, the solution is optimal. The number of instances proven optimal by this test appears as *LP_bound*. However, the linear bound is only applied if the number of remaining variables is fewer than 1,500, to avoid lengthy runs. The sum of the instances proven to be optimal by these three tests appears as *Solved to optimality by preprocessing*. The remaining problems are shown on the last line of the table. For more than 60 % of the non-trivial problems, the preprocessing procedures are able to provide a provably optimal solution.

A characteristic of PROGEN 3 is that it tends to produce large values of the planning horizon T. For this set of problems the reduction of T described in Sect. 11.3 is especially useful. Table 11.4 shows the reduction of T obtained by that procedure on the non-optimally solved problems.

The reductions in the planning horizon T, together with the procedures for reducing possible finishing times for the activities, produce significant decreases in the final number of variables to be used by solution procedures, as can be seen in Table 11.5.

Table 11.3 Schirmer problems—optimal solutions identified in the preprocessing

Activities	10	20	30	40	60
Problems	951	960	960	960	960
Feasible problems	946	960	960	960	960
Trivial problems	143	395	507	574	614
Solved to optimality by pre-processing	502	341	291	221	202
CPM_bound	259	287	279	212	201
MIN_PFT_bound	209	46	10	8	1
LP_bound	34	8	2	1	0
Remaining problems	301	224	162	165	144

Table 11.4 Schirmer problems—reductions of planning horizon T

Activities	10	20	30	40	60
Problems	301	224	162	165	144
Average initial T	43	82	120	158	230
Average reduction	21 %	38 %	43 %	48 %	53 %
Maximal reduction	54 %	66 %	68 %	73 %	73 %

Table 11.5 Schirmer problems—reductions of resources and variables

Activities	10	20	30	40	60
Problems	301	224	162	165	144
Initial resources	30	30	30	30	30
Remaining resources (average)	15 (50 %)	15 (50 %)	18 (60 %)	18 (60 %)	20 (67 %)
Initial variables (average)	211	971	2,281	4,256	9,962
Remaining variables (average)	103 (49 %)	338 (35 %)	718 (31 %)	1,188 (28 %)	2,616 (26 %)

11.6.3 Computational Results of Constructive Algorithms

The 32 priority rules described by Schirmer (2000) were coded and embedded in the constructive algorithm in Sect. 11.4.1. In a preliminary computational study these rules were tested on the 879 feasible instances of size 10 generated by Böttcher et al. (1999). Table 11.6 shows the results obtained by the six best performing rules. The first three rules are based on the network structure of the problems. The last three rules are based on resource consumption. In them, SR indicates that the rules require the use of only scarce resources, indexed by k. RK_{ks} is the remaining capacity of resource k at stage s. RD_{ikt} is the relevant demand, defined as $RD_{ikt} = r_{ik}|Q_{it} \cap \Pi_k|$. MDE_{ikt} is the minimum relevant demand entailed for resource k by all successors of activity i when started at period t. The most important feature of Table 11.6 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.

Table 11.6 Results of priority rules

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

The randomization procedures allowed us to get an important increase in the number of feasible solutions. However, not all these small problems could be solved. That was the reason for the development of a repairing mechanism to help the constructive algorithm to find feasible solutions for the more tightly constrained problems. After this preliminary study we decided to use the second randomization procedure, a biased selection on the Restricted Candidate List, and the priority rule *LFT*.

11.6.4 Computational Results of GRASP Algorithms

Table 11.7 contains the results on the 2,553 non-trivial Schirmer instances, including the 60-activity instances we generated. The first part of the table shows the average distance to optimal solutions. However, not all the optimal solutions are known. The optimal solution is not known for one instance of size 30 and five instances of size 40. In these cases, the comparison is made with the best-known solution, obtained by a time-limited run of the CPLEX integer code, by the best version of GRASP algorithms in any of the preliminary tests or by the best version of the Scatter Search algorithm. The second part of the table 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.

Columns 6 to 9 of Table 11.7 show the results of the GRASP algorithms. Four versions 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, *AG-GR*, the aggressive GRASP procedure, and *AG+PR*, combining aggressive GRASP and Path Relinking. The GRASP algorithms use priority rule *LFT* and the second randomization procedure with $\delta = 0.85$. The GRASP algorithm runs for a maximum of 2,000 iterations while for the aggressive procedure in Sect. 11.4.3 we use the limits described there.

The results in Table 11.7 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 for each instance, but it tends to

Table 11.7 Comparison of Scatter Search and GRASP algorithms on Schirmer problems

Non-trivial		Scatter Search			GRASP			
Activities	instances	Regen 0	Regen 1	Regen 6	GRASP	GR+PR	AG-GR	AG+PR
		Average deviation from optimal solution (%)						
10	803	0.02	0.00	0.00	0.00	0.00	0.00	0.00
20	565	0.15	0.03	0.04	0.40	0.33	0.13	0.12
30	453	0.32	0.19	0.10	1.00	0.88	0.24	0.21
40	386	0.59	0.36	0.25	2.03	1.82	0.67	0.59
60	346	1.22	0.90	0.71	3.68	3.31	1.38	1.16
Total	2,553	0.35	0.04	0.03	0.27	0.24	0.07	0.06
		Non-optimal solutions						
10	803	3	0	0	1	1	1	1
20	565	22	10	5	43	32	22	19
30	453	41	29	21	68	63	35	33
40	386	61	41	31	89	84	59	54
60	346	85	74	67	110	105	91	80
Total	2,553	212	154	124	311	285	208	187
		Average running time						
10	803	1.2	1.8	2.5	0.9	0.9	0.3	0.3
20	565	3.0	3.4	17.0	1.4	1.4	1.1	1.2
30	453	7.2	11.8	28.8	2.9	3.1	3.4	3.7
40	386	15.7	25.8	51.1	5.7	6.2	2.9	7.2
60	346	55.8	105.2	175.9	8.7	10.3	10.6	13.4
Total	2,553	18.3	33.6	60.1	3.6	4.1	4.0	4.9

produce better results especially for large problems. The Path Relinking algorithm adds little improvement to the good results obtained by GRASP procedures.

The last lines of the table provide the running times of the algorithms, in CPU seconds. In all cases preprocessing is 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 average running times are very short, though some problems would require quite long times. In order to avoid excessively long runs, we have imposed a time limit of 60 s for the GRASP procedures. This limit negatively affects the solution of some hard instances, which could have obtained better solutions in longer times, but in general it ensures a good balance between solution quality and required running time. The addition of the Path Relinking procedure does not increase the running times very much, in part because we have included a mechanism to detect solutions which have already been explored and avoid submitting them to the improvement procedure. Therefore it seems convenient to keep it in the final implementation. Therefore, the aggressive GRASP algorithm with Path Relinking seems to be the best option for this type of metaheuristic algorithm.

11.6.5 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 2,000 iterations is reached. From the initial population, a reference set *RefSet* of $b = 10$ solutions is built, with $b_1 = 5$ quality solutions and $b_2 = 5$ diverse solutions.

In a preliminary experience, the eight combination procedures described in Sect. 11.5 have been tested on Schirmer's problems. In this case, no regeneration of the reference set is included and the algorithm stops whenever no new solutions can be added to the Reference Set after a combination phase. We could see that all the combinations except for Combination 7 obtained similar results. So for further testing we keep Combinations 1 and 8, which produce the best results and have completely different structures. We could observe that the basic Scatter Search algorithm is very efficient, obtaining optimal solutions for most of the 3,826 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 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. Maximum number of iterations: 500–1,000
2. Maximum number of new solutions: 20–50
3. Calls to regenerate: three times—Only when the solution is improved after the last call to regenerating

Table 11.7 shows the results for three versions of the Scatter Search algorithm: Regen 0 (without regeneration), Regen 1 (500 iterations, 20 solutions, the regeneration is called while the solution is improved), Regen 6 (1,000 iterations, 50 solutions, regeneration is called three times), and compare them with the best version of the GRASP algorithms. The table shows that while the GRASP algorithm is very efficient and can obtain better solutions on small problems, the Scatter Search procedure with increasingly complex regeneration strategies can significantly improve the overall results with a moderate increase in the running times. Similar results were obtained for the test instances of Böttcher et al. (1999).

11.7 Conclusions

Partially renewable resources, 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, can be seen as a generalization of renewable and non-renewable resources, but their main interest comes from their usefulness for modeling complex situations appearing in timetabling and labor scheduling problems, which can be approached as project scheduling problems.

Preprocessing techniques which help to determine the existence of feasible solutions and to reduce the number of variables and constraints are specially useful for this type of problems because of the existence of a time horizon in which the partially renewable resources are defined.

In this chapter we have reviewed existing formulations and exact algorithms but we have focused on preprocessing techniques and heuristic algorithms, ranging form simple priority rules to sophisticated GRASP/Path Relinking and Scatter Search procedures. The computational results show that these metaheuristic algorithms are able to produce high quality solutions in moderate computing times.

Acknowledgements This work has been partially supported by the Spanish Ministry of Education and Science DPI2011-24977.

References

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2006) GRASP and path relinking for project scheduling under partially renewable resources. Eur J Oper Res 189:1153–1170

Alvarez-Valdes R, Crespo E, Tamarit JM, Villa F (2008) A scatter search algorithm for project scheduling under partially renewable resources. J Heuristics 12:95–113

Böttcher J, Drexl A, Kolisch R, Salewski F (1999) Project scheduling under partially renewable resource constraints. Manage Sci 45:544–559

Chaudhuri S, Walker RA, Mitchell JE (1994) Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem. IEEE T VLSI Syst 2:456–471

Christofides N, Alvarez-Valdes R, Tamarit JM (1987) Project scheduling with resource constraints: a branch and bound approach. Eur J Oper Res 29:262–273

de Boer(1998) Resource-constrained multi-project management: a hierarchical decision support system. Ph.D. dissertation, University of Twente, Twente, The Netherlands

Kolisch R, Sprecher A, Drexl A (1995) Characterization and generation of a general class of resource-constrained project scheduling problems. Manage Sci 41:1693–1703

Laguna M, Marti R (2004) Scatter search. Kluwer, Boston

Mellentien C, Schwindt C, Trautmann N (2004) Scheduling the factory pick-up of new cars. OR Spectr 26:579–601

Neumann K, Schwindt C, Trautmann N (2002) Advanced production scheduling for batch plants in process industries. OR Spectr 24:251–279

Neumann K, Schwindt C, Trautmann N (2005) Scheduling of continuous and discontinuous material flows with intermediate storage restrictions. Eur J Oper Res 165:495–509

Resende, MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenbeger G (eds) Handbook of metaheuristics. Kluwer, Boston, pp 219–249

Schirmer A (2000) Project scheduling with scarce resources. Dr. Kovac, Hamburg

Schwindt C, Trautmann N (2003) Scheduling the production of rolling ingots: industrial context, model and solution method. Int Trans Oper Res 10:547–563

Shewchuk JP, Chang TC (1995) Resource-constrained job scheduling with recyclable resources. Eur J Oper Res 81:364–375

Talbot FB, Patterson JH (1978) An efficient integer programming algorithm with network cuts fo solving resource-constrained project scheduling problems. Manage Sci 24:1163–1174