Available online at www.sciencedirect.com

ScienceDirect

European Journal of Operational Research 189 (2008) 1136–1152

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/ejor

Pre-emption in resource-constrained project scheduling

Francisco Ballestín [a], Vicente Valls [b], Sacramento Quintanilla [c,*]

[a] *Departamento de Estadística e Investigación Operativa, Facultad de Económicas y Empresariales, Universidad Pública de Navarra, Pamplona, Spain*

[b] *Departamento de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Spain*

[c] *Departamento de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Spain*

Received 1 October 2004; accepted 1 July 2006
Available online 26 June 2007

Abstract

The Resource-Constrained Project Scheduling Project (RCPSP), together with some of its extensions, has been widely studied. A fundamental assumption in this basic problem is that activities in progress are non-preemptable. Very little effort has been made to uncover the potential benefits of discrete activity pre-emption, and the papers dealing with this issue have reached the conclusion that it has little effect on project length when constant resource availability levels are defined. In this paper we show how three basic elements of many heuristics for the RCPSP – codification, serial SGS and double justification – can be adapted to deal with interruption. The paper is mainly focussed on problem 1_PRCPSP, a generalization of the RCPSP where a maximum of one interruption per activity is allowed. However, it is also shown how these three elements can be further adapted to deal with more general pre-emptive problems. Computational experiments on the standard j30 and j120 sets support the conclusion that pre-emption does help to decrease project length when compared to the no-interruption case. They also prove the usefulness of the justification in the presence of pre-emption. The justification is a RCPS technique that can be easily incorporated into a wide range of algorithms for the RCPSP, increasing their solution quality – maintaining the number of schedules calculated.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Resource constrained project scheduling; Heuristic algorithms; Pre-emption; Double justification; Serial SGS

1. Introduction

The classical resource constrained project-scheduling problem (RCPSP) may be stated as follows. A project consists of a set of n activities numbered 1 to n, where each activity has to be processed without interruption to complete the project. We consider additional activities $j = 0$ and $j = n + 1$ representing the single source and single sink activity of the network respectively. The duration of an activity j is denoted by p_j, where $p_0 = p_{n+1} = 0$. There are K renewable resource types. The availability of each resource type k in each time period is R_k units, $k = 1, \ldots, K$. Each activity j requires r_{jk} units of resource k during each period of its duration where $r_{0k} = r_{n+1,k} = 0$, $k = 1, \ldots, K$. All parameters are assumed to be non-negative integer valued. There

[*] Corresponding author. Tel.: +96 3828396; fax: +96 3828370.
E-mail addresses: Francisco.Ballestin@uv.es (F. Ballestín), Vicente.Valls@uv.es (V. Valls), Maria.Quintanilla@uv.es (S. Quintanilla).

0377-2217/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.07.052

are precedence relations of the finish-start type with a zero parameter value (i.e., $FS = 0$) defined between the activities. In other words, activity i precedes activity j if j cannot start until i has been completed. The structure of a project can be represented by an activity-on-node network $G = (V, A)$, where V is the set of activities and A is the set of precedence relationships. $S_j(P_j)$ is the set of successors (predecessors) of activity j. It is assumed that $0 \in P_j$, $j = 1, \ldots, n+1$, and $n+1 \in S_j$, $j = 0, \ldots, n$. The objective of the RCPSP is to find a schedule S of the activities, i.e., a set of starting times $(s_0, s_1, \ldots, s_{n+1})$, where $s_0 = 0$ and the precedence and resource constraints are satisfied, such that the schedule duration $T(S) = s_{n+1}$ is minimised.

In recent years, the applications and difficulties of the RCPSP and its extensions have attracted increasing interest from researchers and practitioners. We refer to the surveys published by Icmeli et al. (1993), Özdamar and Ulusoy (1995), Kolisch and Padman (2000), Herroelen et al. (1998), Brucker et al. (1999) and the books by Weglarz (1999), Demeulemeester and Herroelen (2002), Neumann et al. (2003). One of the extensions of the RCPSP that has received little attention from the researches has been the RCPSP with pre-emption, i.e., when activities are allowed to be interrupted. This is due to the often unjustified assumption that an activity which has been started cannot be interrupted before its completion. In fact, all main project management software packages include pre-emptability among their capabilities.

There are, however, some papers concerning pre-emption and other project scheduling problems – such as when objective functions other than the project makespan are considered (Bock and Patterson, 1990), or when multiple projects are considered (Tsubakitani and Deckro, 1990). In this paper, we deal with pre-emption in the classical RCPSP, specifically when integer processing times are considered, in contrast to when continuous processing times are assumed (see Slowinski, 1980; Weglarz, 1981). We will also consider the suggestion that interrupting an activity has a negligible cost.

Kaplan (1988, 1991) was the first to study the pre-emptive resource-constrained project scheduling problem (PRCPSP). The PRCPSP can be defined as the RCPSP where the non-preemption assumption is relaxed. In the PRCPSP, activities are allowed to be pre-empted at any integer time instant and restarted later on at no additional cost, i.e., each activity can be split into, at most, p_i parts of one duration unit each. Kaplan provided a procedure for the PRCPSP based on a dynamic programming formulation. Demeulemeester and Herroelen (1996) proved that the solution procedure proposed by Kaplan was wrong and developed a correct branch-and-bound procedure for the PRCPSP.

Demeulemeester and Herroelen tested their algorithm on the well-known Patterson instance set (Patterson, 1984). They noticed a 33-fold increase in the average computational time when compared to that of their exact algorithm (Demeulemeester and Herroelen, 1992) for the (non-preemptive) RCPSP. However, a very small decrease in the optimal project lengths was found; for 30 of the 110 problems the optimal project length for the pre-emptive case was one time unit smaller than that for the non-preemptive case, in all other cases both optimal project lengths were identical. These results made them assert that in project scheduling the introduction of pre-emption has little effect when constant resource availability levels are defined, a conclusion also held by Kaplan.

However, it has been pointed out (Kolisch et al., 1995) that the Patterson set suffers from two main drawbacks: (a) as a collection of instances from different sources, the instances are not generated by using a controlled design of specified parameters, and (b) there are instances (with the same number of activities) which are much more difficult to solve. For these reasons, they should not be used as a benchmark. These disadvantages of the Patterson set put in doubt the conclusions obtained in the aforementioned papers. One of the purposes of this paper is to re-evaluate the usefulness of pre-emption for decreasing the project length in the RCPSP.

The experiments of Kaplan and Demeulemeester and Herroelen were conducted using exact algorithms. However, as a job shop generalisation, the RCPSP is NP-hard in the strong sense (see Blazewicz et al., 1983). Accordingly, we have to rely on heuristic procedures for solving RCPSP instances of practical sizes. That is why in this paper we have adopted a practically oriented approach in the sense that we want to ascertain whether the addition of the capability of pre-emption to heuristic algorithms for the RCPSP is useful for decreasing the makespan of the generated schedules.

Although allowing activity interruption may reduce the duration of a project, the repeated stopping and starting of an activity may not be managerially admissible and several authors, e.g. Crandall (1973), have proposed limiting the number of interruptions to a maximum of one per activity. This practical reason is backed up by the work of Lino (1997). She studied the effect of pre-emption on the problem of minimizing the makespan of a project with generalised precedence relationships with minimal time-lags but without any consideration to resource constraints. As far as the process of the activities is concerned she considered three settings: no interruption, any number of interruptions (at integer time instants) and a maximum of one interruption per activity (at an integer time instant). She concluded that if each activity is allowed to be interrupted just once, then a significant reduction in project length is obtained when compared to the case of no interruption. Also, she detected that allowing more than one interruption instead of a maximum of one interruption per activity does not further reduce the project length in the majority of the instances – and that when a reduction happens it is very small. For this reason, she also concluded that the benefits of allowing more than one interruption per activity may not compensate in practice. These conclusions were drawn using a test bed of 1500 randomly generated instances ranging up to 1000 activities and 12,000 generalised precedence relationships.

In this paper we adhere to that practical point of view: we study the effect of pre-emption on project makespan – but limiting the number of interruptions to a maximum of one per activity. In the next section, we give further and more technical motivations for adopting this constraint.

Valls et al. (2005) showed that justification – specifically double justification – is a simple and quick technique that can be incorporated into many diverse algorithms for the RCPSP and produces significant improvements in the quality of the sequences generated. The improvement in the quality of 22 tested algorithms was obtained in most cases without increasing computing time and in some cases it greatly reduced computing time. Based on the obtained results, the authors conclude that the double justification should always be considered when designing an algorithm for the RCPSP. In fact, the best current heuristics for the RCPSP use justification (Valls et al., 2002). In this paper we will show that the advantages of the justification are also valid when pre-emption is permitted.

The activity list representation of schedules and the serial schedule generation scheme (Hartmann, 1998) are two basic procedures that are not specific to a particular type of algorithm; on the contrary, they are extensively used by many heuristic algorithms for the RCPSP. Another goal of this paper is to show how these three basic widely-used techniques can be adapted to deal with pre-emption.

The rest of the paper will be as follows. In Section 2, we will define the pre-emption problem we are going to work with, as well as discuss the reasons for this choice. Section 3 explains the adjustments made to transform algorithms for the RCPSP into algorithms for the pre-emption problem 1_PRCPSP. This includes the codification of a solution, the decoding procedure and the justification. Section 4 shows how these three key algorithmic elements can be further adapted to more general pre-emption problems. Specifically, the pre-emption problem m_PRCPSP will be considered. Section 5 describes the algorithms used to prove the differences in project length of the solutions obtained with the pre-emption problem and the classical RCPSP. The computational results concerning these algorithms are shown in Section 6, and Section 7 is reserved for concluding remarks.

2. Problem 1_PRCPSP

The *m*_PRCPSP, *m* integer, can be defined as the RCPSP where the non pre-emption assumption is relaxed in the following way: activities are allowed to be pre-empted a maximum of m times at any integer time instant and restarted later on at no additional cost. Clearly, *i*_PRCPSP is a generalization of *j*_PRCPSP for $i > j$; 0_PRCPSP is the same as RCPSP and for each instance there exists an integer m_0 such that to solve problem PRCPSP is equivalent to solve problem *i*_PRCPSP for $i \geqslant m_0$, where m_0 is the maximum activity duration minus one. This paper is mainly focussed on the 1_PRCPSP.

Let us consider the project given in Fig. 1. The schedule *S* depicted in Fig. 2 is feasible and optimal for the PRCPSP and also for the i_PRCPSP for $i \geqslant 2$. However, it is not feasible for the 1_PRCPSP, as activity 1 is interrupted twice. Note

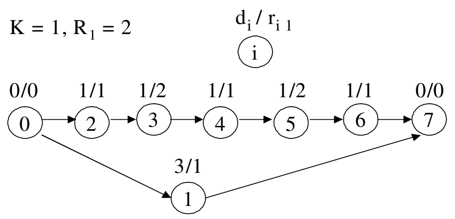

Fig. 1. Project 1.

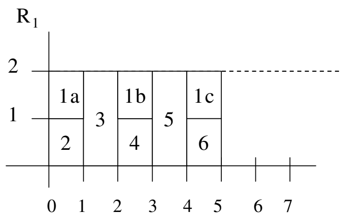

Fig. 2. An optimal PRCPSP schedult (1).

that we have denoted the first, second and third parts of activity 1 as 1a, 1b and 1c, respectively. A feasible and optimal solution for the 1_PRCPSP, one unit longer than S, is depicted in Fig. 3 (1a and 1b are again the first and second parts of activity 1). One optimal solution for the RCPSP is given in Fig. 4 and is one unit longer than the one for the 1_PRCPSP.

Apart from the managerial reason exposed in the introduction, there are further attractive reasons for studying problem 1_PRCPSP. There is an easy way of solving problem PRCPSP using existing RCPSP algorithms. It is only necessary to apply the selected algorithm to a new project obtained from the original project by splitting each activity i into p_i parts of one duration unit each; connecting them with precedence relationships and assigning them the same resource requirement as i. In this way, high quality solutions for this problem can be presumably obtained if state-of-the-art heuristic algorithms for the RCPSP are applied. However, it is not possible to design a similar procedure for solving the 1_PRCPSP because prior to the construction of the new project it would be necessary to decide how to split the activities.

Furthermore, the knowledge gained studying problem 1_PRCPSP can be extended to its generalizations. Specifically, the makespan reduction obtained by allowing at most one interruption per activity is a lower bound of the reduction obtainable by allowing at most $m \geqslant 2$ interruptions per activity. Also, as it will be commented upon with more extension in Section 3, the procedures for solving 1_PRCPSP we have developed in this paper can be easily adapted to deal with more general cases.

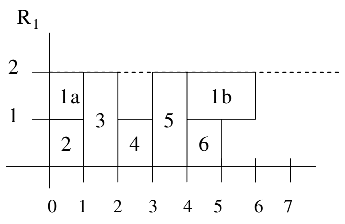

Fig. 3. An optimal 1_PRCPSP schedule (1).

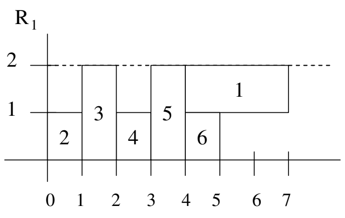

Fig. 4. An optimal RCPSP schedult (1).

3. Adapting some basic procedures

The codification of the solutions of a problem and the manner to decode them is essential for the construction of heuristic algorithms. One of the most important types of codification of schedules in the RCPSP is the activity list. An activity list $\lambda = (j_1, \ldots, j_n)$ is a precedence feasible permutation of the activities. To shorten notation we omit activities 0 and $n + 1$ without risk of confusion.

The usual way of transforming (decoding) an activity list into a schedule in the RCPSP is by means of the serial schedule generation scheme (SGS). The serial SGS builds schedules from activity lists as follows. First, the dummy source activity is started at time 0. Then the activities are scheduled in the order prescribed by the list $(j_1 \ldots j_n)$. Thereby, each activity is assigned the earliest precedence and resource feasible start time. Finally, the dummy sink activity is started and finished at time $T(S) = \max\{s_i + p_i, i = 0, \ldots, n\}$.

The serial SGS produces the so-called active schedules (cf. Sprecher et al., 1995), where each activity different from 0 cannot be scheduled earlier without changing the schedule period of other activities. The serial SGS is capable of generating every active schedule, thus being able to find an optimal solution for each feasible instance, as the set of active schedules always contains an optimal schedule. Retaking the project depicted in Fig. 1, the serial SGS transforms $\lambda = (2\,3\,4\,5\,6\,1)$ into the schedule of Fig. 4.

The justification of a schedule in RCPSP was introduced in 1964 by Wiest but has not been used to its full potential until recently. It is a very easy and fast procedure that never increases the makespan of a schedule and in many cases shortens it. The justification of a schedule S to the right (left) consists in fixing the dummy activity $n+1$ (0) in $t = T(S)$ $(t = 0)$ and scheduling each activity as late (early) as possible in decreasing (increasing) order of their ends (beginnings).

Valls et al. (2005) have used the justification in a very concrete and effective way, by terms of the double justification or DJ. Two of the best current heuristic algorithms include the DJ as one of their most important features (cf. Valls et al., 2002, 2005). The DJ of a schedule S consists in justifying it to the right and then justifying the resulting schedule to the left. The application of the DJ to a schedule S always yields an active schedule, DJ(S). Since many heuristic algorithms work with active schedules, it is very easy to add the DJ to them. One only needs to apply the DJ to every schedule generated and work with the resulting schedule instead of the original schedule.

The wise adaptation of these three techniques is a fundamental step prior to transforming algorithms for the RCPSP into algorithms for the 1_PRCPSP. This is discussed in the following sub-sections.

3.1. Codification of a solution in the 1_PRCPSP

In the 1_PRCPSP, any activity may be divided into a maximum of two parts. So, a schedule in the 1_PRCPSP consists of specifying for each activity whether it has been interrupted or not, and the starting times and the duration of the two parts – or those of the whole activity, depending on the case. We propose to codify the schedules of problem 1_PRCPSP by means of two vectors: a vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_{2n})$ that we will call a double activity list and a first-part duration vector $\mathbf{d} = (d_1, d_2, \ldots, d_n)$. The double activity list λ contains each activity twice, which can appear in λ in any position after all its predecessors. The first (second) appearance of an activity i in λ, which we will call **ia** (**ib**), represents the first (second) part of the interrupted activity i. The ith position in $d, d(i)$, stores the duration of the first part of i, which we will denote d_{ia}. The duration of ib (denoted as d_{ib}) can be calculated as $p_{\text{i}} - d_{\text{ia}}$. The case when an activity i is not interrupted is codified with $d_{\text{ia}} = p_{\text{i}}$, and ib becomes a sort of dummy activity. Later on we will offer some examples.

Note that any double activity list for a project G with n non-dummy activities can be considered as an activity list for a project G′ with non-dummy $2n$ activities where G′is built from G as follows:

1. For each non-dummy activity $i, i = 1, \ldots, n$, consider two activities ia and ib. Consider also the dummy activities 0 and $2n+1$.
2. For each arc (i,j) in G, $i \neq 0$ and $j \neq n+1$, consider arc (ib, ja).
   For each arc $(0,i)$ in G, consider arc (0, ia).
   For each arc $(i, n+1)$ in G, consider arc (ib,$2n+1$).
   For each non-dummy activity i consider arc (ia, ib).

We define a **representation** of a schedule S as a pair, (λ, d), of a double activity list and a first-part duration vector with the following two features: The double activity list λ is a permutation of the parts of the activities in non-decreasing order of their starting times in S. Uninterrupted activities i are codified by two appearances of activity i one immediately after the other. The first-part duration vector d stores the duration of the first parts of the activities in S. The algorithms presented in Section 5 works with representations of schedules.

3.2. The serial SGS

Given a pair (λ, d) of a double activity list and a first-part duration vector, the serial SGS can be used to obtain a precedence and resource feasible schedule for the 1_PRCPSP. First, the dummy source activity is started at time 0. Then the parts of the activities are scheduled in the order prescribed by the double activity list λ. The first time an activity i appears in λ represents the first part of activity i with a duration of $d(i)$ time units. The second appearance of i represents the second part of activity i with a duration of $p_i - d(i)$ time units. The first part of an activity is considered as a predecessor of its second part. Each part of an activity is assigned the earliest precedence and resource feasible start time, as in the RCPSP. If $p_i = d(i)$ then the second part should be considered as a dummy activity.

Note that if S is the schedule obtained by the application of the serial SGS to the pair (λ, d) and (λ', d') is a representation of S then the serial SGS applied to (λ', d') yields S. Note also that $d' = d$ although λ' is not necessarily equal to λ. Similarly to the non pre-emptive case, it is not difficult to show that the serial SGS for the 1_PRCPSP is able to find an optimal schedule for every feasible 1_PRCPSP instance. Recall that a double activity list for a project G can be considered as an activity list λ' for a project G′. Note also that to apply the serial SGS for the 1_RCPSP to λ is equivalent to apply the serial SGS to λ'.

3.3. The 1_serial SGS

The above described serial SGS for the 1_PRCPSP is a straightforward adaptation of the serial SGS for the RCPSP to problem 1_PRCPSP. However, we can take advantage of the characteristics of problem 1_PRCPSP to improve it. Given a pair (λ, d) the new procedure, which will be called 1_serial SGS, proceeds as the serial SGS for the 1_PRCPSP except for:

(a) When dealing with the first part of an activity i with a dummy second part (i.e., when $p_i = d(i)$), it looks for the first instant, t, where all predecessors of i have finished and at least one unit of i can be scheduled. Then it calculates the maximum number, q, of units of i that can be scheduled without interruption at time t. If $q > p_i$, i can be completely scheduled in $[t, t + p_i]$, otherwise the new procedure interrupts i, and schedules the first part of q units in $[t, t + q]$ and the second part as early as possible without the possibility of splitting it again.

(b) When the 1_serial SGS deals with the second part of an activity i with $p_i - d(i) > 0$, it schedules as many units as possible (can be 0) immediately after the end of the first part. The remainder, without the possibility of interruption, is scheduled as early as possible.

(c) In the case of (b), if the second part has been completely scheduled immediately after the first part, the procedure process activity i again as in case (a).

A formal definition of the procedure 1_serial SGS is shown in Fig. 5.

3.4. Serial SGS versus 1_serial SGS

Note that if S is the schedule obtained by the application of the 1_serial SGS to the pair (λ, d) and (λ', d') is a representation of S then the 1_serial SGS applied to λ', d' yields S. Note also that neither d' nor λ' are necessarily equal to d and λ, respectively. Furthermore, contrary to the serial SGS, the 1_serial SGS is able to create new interruptions in a schedule. This property facilitates the transformation of RCPSP heuristics into 1_PRCPSP heuristics: there is no need for additional mechanisms to produce interruptions.

The following examples illustrate the differences between both adaptations:

(a) The serial SGS transforms (λ, d) with $\lambda =$(1a 1b 2a 3a 3b 2b 4a 4b 5a 5b) and $d =$ (2 2 1 4 1) into the schedule in Fig. 7, feasible for the project in Fig. 6. The 1_serial SGS can split activity 4 and obtains the schedule in Fig. 8, 2 units better.

(b) The serial SGS applied to (λ, d) with $\lambda =$ (1a 1b 4a 4b 2a 5a 5b 3a 3b 2b 6a 6b) and $d =$ (2 2 1 4 1 1) leads to the schedule in Fig. 10, feasible for the project in Fig. 9, while the 1_serial SGS obtains a one unit better schedule in Fig. 11, because it is able to change the duration of 2a from 2 to 3.

(c) The schedule in Fig. 13 is feasible for the project in Fig. 12. It is calculated by means of the serial SGS applied to (λ, d) with $\lambda =$ (1a 1b 4a 3a 3b 4b 2a 5a 5b 6a 6b 2b) and $d =$ (2 3 1 2 1 1). The 1_serial SGS, however, obtains the schedule in Fig. 14, 1 unit shorter because the procedure is able to interrupt the join of activity 2a and 2b.

There are also examples where the serial SGS can lead to a better solution than the 1_serial SGS. We consider the project of Fig. 15, $\lambda =$ (1a 1b 2a 2b 3a 3b 4a 4b 5a 5b) and $d =$ (2 1 3 2 2). The serial SGS transforms (λ, d) in the schedule S of Fig. 17, with 6 units of makespan. The 1_serial SGS obtains the schedule of Fig. 16, one unit longer than S, when applied to the same (λ, d). This unwanted result cannot occur if (λ, d) is a representation of a schedule. Let (λ, d) be a representation of a schedule S. As we have already mentioned the application of the serial SGS to (λ, d) yields S. Let S' be the schedule obtained by applying the 1_serial SGS to the pair (λ, d). It is clear that $s'_i \leqslant s_i$, where s'_i is the beginning of activity i in S'. So, S' is always as good as S, but sometimes it can be shorter. In this sense, the 1_serial SGS is better than the serial SGS if both procedures are applied to schedule representations – as is the case with the algorithms presented in Section 5. From the above reasoning, we can also conclude that the 1_serial SGS is capable of obtaining an optimal solution for every instance in the 1_PRCPSP (Fig. 18).

```
$f_{0b}=0$
For h=1,…,2n:
  1. if $\lambda_h = ia$ for some activity i, then:
     $ES_i = \max\{f_{jb}, j \in P(i)\}$
     1.1. if $d_{ia} < p_i$, then
          $t^* = \min\{t \geq ES_i, R_k(\tau) + r_{ik} \leq R_k$ for $t \leq \tau \leq t + d_{ia}$ and $k=1,\dots,K\}$.
          Schedule the first $d_{ia}$ units of activity i at t*: $s_{ia}=t^*$, $f_{ia} = s_{ia} + d_{ia}$.
     1.2. if $d_{ia} = p_i$, then:
          $t^* = \min\{t \geq ES_i, R_k(t) + r_{ik} \leq R_k$ for $k=1,\dots,K\}$
          $u = \max\{\tau \leq p_i - 1 \ / \ R_k(t^*+\tau) + r_{ik} \leq R_k$ for $k=1,\dots,K\}$
          if $u = p_i - 1$ then:
              Schedule $p_i$ units of activity i at t*: $s_{ia} = t^*$, $f_{ia} = s_{ia} + p_i$, $f_{ib} = f_{ia}$, $s_{ib} = f_{ib}$.
          else,
              Schedule 1+u units of activity i at t*: $s_{ia} = t^*$, $f_{ia} = s_{ia}+1+u$, and the remaining
              units as soon as possible without further interruption: $f_{ib} = s_{ib}+p_i-1-u$.
  2. if $\lambda_h = ib$ for some activity i, with $d_{ib} > 0$:
     $u = \max\{\tau \leq d_{ib} \ / \ R_k(s_{ia}+d_{ia}-1+\tau) + r_{ik} \leq R_k$ for $k=1,\dots,K\}$
     2.1. if $u=d_{ib}$ then:
          Unschedule ia: consider $\lambda_h = ia$ with $d_{ia}=p_i$ and proceed as in 1.2.
     2.2. if $0<u<d_{ib}$, then,
          Schedule u units of activity i at $s_{ia} + d_{ia}$, $f_{ia} = f_{ia}+u$, and the remaining $d_{ib} - u$ units as
          soon as possible without further interruption: $f_{ib} = s_{ib}+d_{ib}-u$.
     2.3. if u=0 then,
          Schedule $d_{ib}$ units of activity i as soon as possible without further interruption: $f_{ib} =$
          $s_{ib}+d_{ib}$.
Where:
     $A(t) = \{j \ / \ s_{ja} \leq t < f_{ja}$ or $s_{jb} \leq t < f_{jb}\}$     (set of activities in progress at time t)
     $R_k(t) = \sum_{j \in A(t)} r_{jk}$, $k=1,\dots,K$     (number of type k resources used at time t)
     $f_i$ : finishing times.
```

Fig. 5. 1_serial SGS procedure.

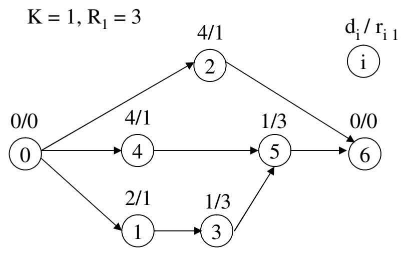

Fig. 6. Project 2.

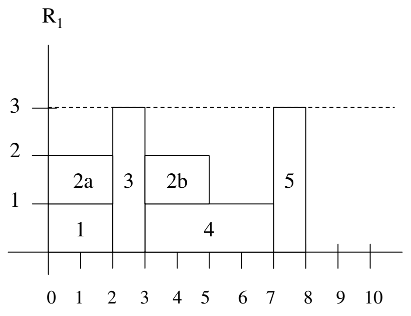

Fig. 7. Serial SGS (2).

3.5. The justification in the 1_PRCPSP

It has been proven in the context of the RCPSP (Ballestín, 2002) that justifying a schedule *S* to the left is equivalent to applying the serial SGS to the activity list formed with the activities in non-decreasing order of their starting times. Analogously, justifying a schedule *S* to the right is

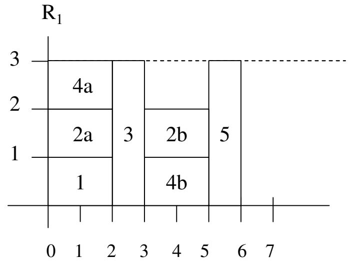

Fig. 8. 1_Serial SGS (2).

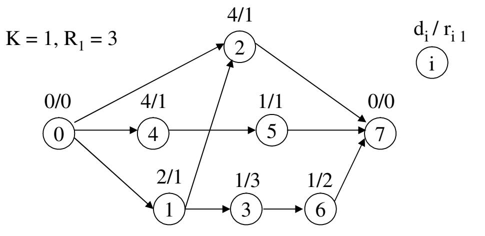

Fig. 9. Project 3.

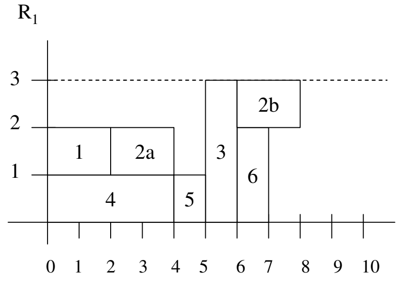

Fig. 10. Serial SGS (3).

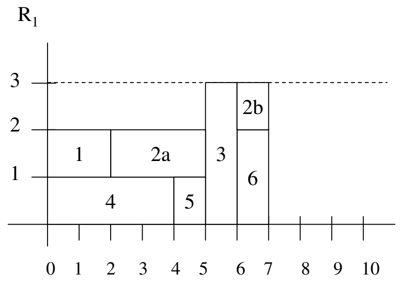

Fig. 11. 1_Serial SGS (3).

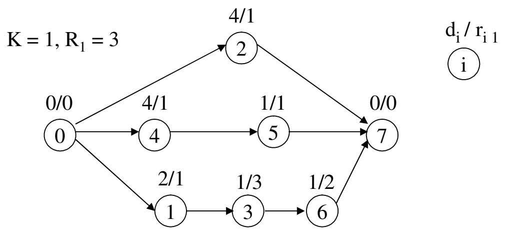

Fig. 12. Project 4.

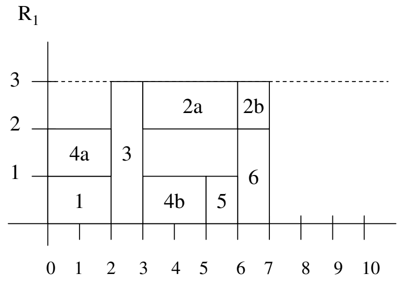

Fig. 13. Serial SGS (4).

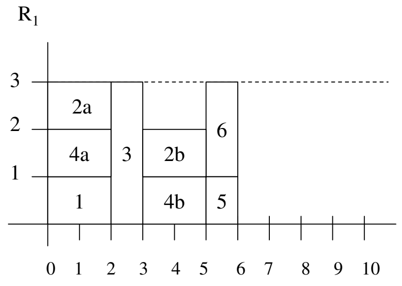

Fig. 14. 1_Serial SGS (3).

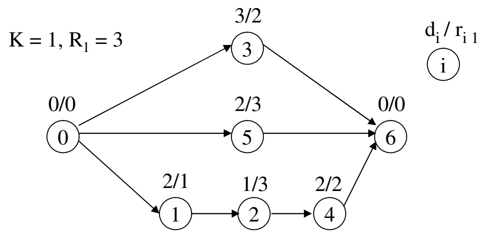

Fig. 15. Project 5.

equivalent to applying the serial SGS to the activity list for the reverse project formed by the activities in non-increasing order of their finishing times in S and obtaining a schedule S'. And then, the beginning of activity i is calculated by $T(S) - f_i'$, where f_i' is the end of activity i in S'. The **reverse project** is the network obtained by simply reversing the precedence relationships of the original project.

According to the aforementioned equivalences, applying the DJ to a schedule S is the same as twice applying the serial SGS. Firstly, it is applied in the reverse project to the activity list formed with the activities in non-increasing order of their finishing

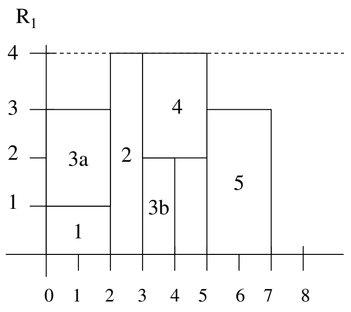

Fig. 16. 1_Serial SGS (5).

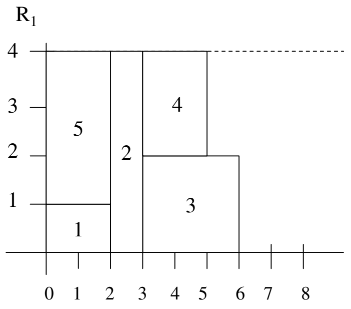

Fig. 17. Serial SGS (5).

times in S, obtaining S'. Secondly, it is applied in the original project, to the activity list formed with the activities in non-decreasing order of their starting times in S'.

Taking this into account, it is straightforward to adapt the DJ to the 1_PRCPSP. It is sufficient to apply the 1_serial SGS twice, firstly in the reverse project and, secondly in the original project. To be more precise, let S be a schedule in the 1_PRCPSP. We define a **right representation** of a schedule S as a pair, (λ, d), of a double activity list in the reverse project and a first-part duration vector with the following two features. The double activity list λ is a permutation of the parts of the activities in non-increasing order of their finishing times in S. Uninterrupted activities i are codified by two appearances of activity i, one immediately after the other. The first-part duration vector d stores the duration of the second parts of the activities in S. The adaptation of the DJ procedure can be more formally described as follows:

Therefore, once the 1_serial SGS is programmed only a few lines of code are needed to implement the DJ. Notice that DJ for the 1_PRCPSP can modify the interruptions in a schedule thus adding further flexibility to any procedure that incorporates it. Figs. 19–22 illustrate the procedure 1_DJ(S), where
$\lambda = (5a,5b,4a,4b,2a,3a,3b,2b,1a,1b)$,
$d = (1, 2, 1, 5, 1)$,
$\lambda'' = (3a,3b,4a,4b,2a,2b,1a,1b,5a,5b)$
$d'' = (1, 4, 1, 5, 1)$ (Fig. 23).

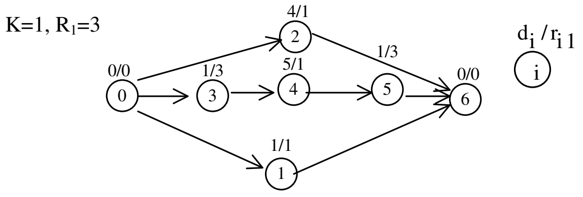

Fig. 19. Project 6.

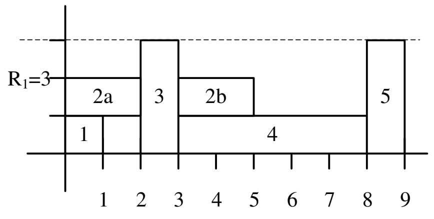

Fig. 20. S.

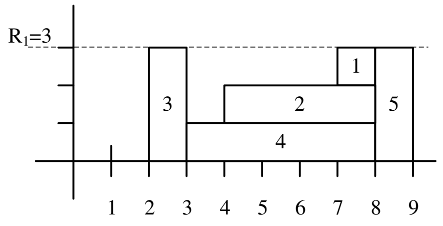

Fig. 21. S'.

1. Let (λ,d) be the right representation of S.
2. Apply 1_serial SGS to (λ,d) in the reverse project and obtain S'.
3. Let S'' be the schedule obtained from S' by subtracting s'_0 from all starting times s'_h.
4. Let (λ'',d'') be the representation of S''.
5. Apply 1_serial SGS to (λ'',d'') in the original project and obtain S'''.
6. Return S'''.

Fig. 18. 1_DJ(S) procedure.

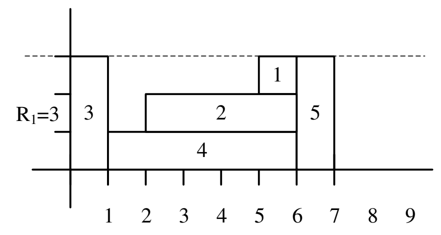

Fig. 22. S''.

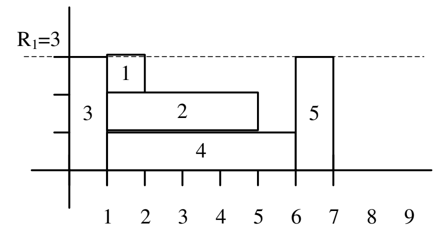

Fig. 23. DJ(S).

4. Adaptation to more general pre-emption problems

In the previous section, we have adapted three key RCPSP algorithmic elements – codification, SGS and justification – to the 1_PRCPSP. However, these adaptations can be further adapted to more general pre-emption problems. For example, we can consider pre-emption project scheduling problems that include some or all the following constraints: the maximum number of allowed interruptions is $m \in Z^+$ for all activities (m_PRCPSP); the maximum number of allowed interruptions is activity dependant, possibly zero for some activities; the duration of the parts in which an activity can be split cannot be less than a minimum value. We will show how the three key algorithmic elements – codification, serial SGS and justification – can be further adapted to deal with these types of constraints. Specifically, we will consider the problem that can be defined as a generalization of the RCPSP where each activity i can be interrupted a maximum of n_int(i) times at any integer instant and restart later on at no additional cost. This should be done in such a way that the parts in which an activity can be split must be of duration min_dur(i) as a minimum. We define $m = \max\{n\_int(i) + 1, i \in V\}$.

4.1. Codification of a solution

We propose to codify the schedules of this problem by means of $m + 1$ vectors: a vector λ of $m \cdot n$ positions that we will call m-activity list and m vectors $d^1, d^2, \ldots, d^m$ of n cells each. The m-activity list λ contains each activity m times, which can appear in λ in any position after all its predecessors. Each appearance of an activity i in λ, i_j, represents the jth part of the (possibly interrupted) activity i. The ith position in d^jstores the duration, $d^j(i)$, of the jth part of $i, j = 1, \ldots, m$. The case when an activity i has been partitioned into $g < m$ parts is codified with $d^j(i) = 0$, $j = g + 1, g + 2, \ldots, m$ and all these i_j become a sort of dummy activities. Notice that to be feasible a codification must fulfil the restrictions imposed by n_int(i) and min_dur(i) for $i \in V$.

In a representation of a schedule S, the m-activity list λ would be a permutation of the parts of the activities in non-decreasing order of their starting times in S. Dummy parts i_j would appear immediately after the last non-dummy part i_g.

4.2. Adapting the serial SGS

Given a m-activity list and $d^1, d^2, \ldots, d^m$, the serial SGS could be used to obtain a precedence and resource feasible schedule for the problem, handling each non-dummy part as a whole activity. However, as we have done in the case of the 1-PRCPSP, we can take advantage of the characteristics of the problem to improve the scheduling scheme. We would only need to adapt the changes we have made in the 1-serial SGS to the case of more interruptions. Given a codification of a schedule, the new procedure proceeds as the serial SGS but with the following changes, where curr_int(i) is the current number of interruptions of activity i in the partial schedule, taking also into account the parts not yet scheduled.

(a) When dealing with the first part of an activity i, if curr_int(i) = n_int(i) the procedure schedules it as early as possible. If curr_int(i) < n_int(i), the procedure tries to finish this part earlier by separating it and scheduling it in two (or more, if feasible) parts.

(b) When the procedure deals with the jth part of an activity i, $j > 1$, with $d^j(i) > 0$, it schedules as many units as possible (can be 0) immediately after the end of the $j - 1$th part. If curr_int(i) = n_int(i), the remainder is scheduled as early as possible without the possibility of interruption. If curr_int(i) < n_int(i), the procedure tries to finish the remainder earlier by separating it and scheduling it in two (or more, if feasible) parts.

(c) In the case of (b), if the jth part has been completely scheduled immediately after the $j-1$th part, the procedure processes the whole $j-1$th part again, taking into account that now $\text{curr\_int}(i) = \text{curr\_int}(i) - 1$.

Note that each time an activity is separated, the restrictions imposed by min_dur(i) must be observed.

The adaptation of the double justification is now straightforward because applying the DJ to a schedule S is the same as applying the serial SGS twice.

5. The experiment

5.1. Adapting RCPSP heuristics to the 1_RCPSP

In Section 3, we have seen how three basic elements of many heuristic algorithms for the RCPSP can be adapted to the 1_PRCPSP. In this section, we are going to show how heuristic algorithms for the RCPSP based on activity list, or equivalent, codifications of schedules can be easily adapted to the 1_PRCPSP by just using the three basic adaptations mentioned above.

To adapt the algorithms we will follow the following strategy:

(a) Use double activity lists and first-part duration vectors instead of activity lists.
(b) Apply the 1_serial SGS instead of the serial SGS.
(c) Apply the DJ for the 1_PRCPSP instead of the DJ for the RCPSP.
(d) To take full advantage of the 1_serial SGS, the adapted algorithms will work only with representations of schedules. To this end, any pair generated by the algorithm is replaced by a representation of the schedule obtained by applying the 1_serial SGS to the original pair.
(e) Heuristic algorithms for the RCPSP based on activity list codifications include (original) procedures for generating and/or transforming activity lists. Given that a double activity list for a project can be considered as a double activity list for another project, the original procedures can be applied to the case of double activity lists. Therefore, to completely specify an adapted procedure for the 1_PRCPSP it is only required to define a procedure for generating and/or transforming first-part duration vectors. However, there is usually a 'natural' way of doing it. As an example, let us consider two cases that are relevant to the computational experiment described in Section 5. The first case is the following. If we apply any of the biased random sampling methods devised for the RCPSP (Kolisch, 1995) to generate a double activity list, it seems 'natural' to randomly generate the durations of the first parts of the activities. Another case is that of the two-point crossover used by Hartmann (1998), for crossing two activity lists to generate a new activity list. In this case, the Hartmann crossover operator can be adapted for crossing two double activity lists, (λ, d) and (λ', d'), to obtain a new double activity list (λ'', d'') in the following way. First, apply the Hartmann crossover to λ and λ', and obtain λ''. Then define $d''(i)$ as either $d(i)$ or $d'(i)$ depending on whether the position of ia in λ'' has been inherited from either λ or λ'.

This strategy can be termed as a minimum change strategy. Although, at least theoretically, better results could be obtained with a more disruptive strategy, we have adopted this strategy because we wanted to show that it is possible to reduce the best makespan obtained by a heuristic for the RCPSP, just by allowing pre-emption and without significantly changing the procedure – simply by changing the codification, the serial SGS and the double justification.

5.2. The tested algorithms

In the RCPSP, it is a common practice (see, for example, Hartmann and Kolisch, 2000) to compare heuristic algorithms by comparing the quality of the best schedules obtained by the algorithms when the maximum number of schedules that can be generated is limited to a maximum of 5000. This practice allows the comparison of results obtained using different computers. The rationality being that algorithms with similar encoding and decoding procedures would spend similar computation times if they were run on the same computer. Following this practice, all algorithms tested in this study will be allowed to calculate 5000 schedules at most.

When running a heuristic algorithm for the RCPSP based on activity lists the main part of the computational time is usually devoted to the application of the serial SGS. Specifically, to schedule an activity it is necessary to search along the time axis until an interval with enough available resources is found. When the 1_serial SGS is applied to a pair (λ, d), it has to allocate p_i units of activity i for every activity i (either in one or in two parts). However, to allocate the p_i units of i in two separate (and predefined) parts will usually need less computational time than in one part, since you do not have to find a large enough interval with sufficient resources. This counterbalances the time needed by those changes we introduced in the serial SGS to obtain the 1_serial SGS. This justifies our decision to take the number of generated schedules as the basis for the comparison of all the algorithms considered in this paper.

In the experiment we have considered four algorithms for the RCPSP: two algorithms that do not use justification (Random and GA) and the same algorithms after the incorporation of double justification (DJRandom and DJGA). We have also considered their minimum change strategy adaptations to the 1_PRCPSP (1_Random, 1_DJRandom, 1_GA, and 1_DJGA).

Algorithm Random (1_Random) calculates 5000 random activity lists (double activity lists); applies the serial (1_serial) SGS to them and returns the best solution obtained. The average difference between the project lengths obtained by Random and 1_Random is an estimation of the difference between the average quality of the (active) schedules in the sets of feasible solutions to the RCPSP and to the 1_PRCPSP.

Algorithm DJGA (cf. Valls et al., 2005) is a double justification version of the genetic programming activity list algorithm of Hartmann, GA, (Hartmann, 1998), which is based on the activity list codification and the crossover of two points for permutations. GA has been considered up to a few years ago as one of the best heuristic algorithms for the RCPSP (cf. Hartmann and Kolisch, 2000). Besides, it is still one of the fastest algorithms for calculating 5000 schedules. The incorporation of DJ in this algorithm produced a great increase in the solution quality while maintaining the same number of schedules calculated (5000). Furthermore, DJGA needs approximately the same computational time as GA (a difference of less than 0.1 seconds on average). Both features make DJGA one of the best current heuristic algorithms with a limit of 5000 schedules. This means that being able to obtain better quality solutions than it if pre-emption is allowed will entail obtaining better quality solutions than almost all the many existing heuristic algorithms for the RCPSP.

It is worth noting that the adaptation of DJGA to 1_DJGA is straightforward once the 1_serial SGS and the DJ procedure have been programmed. It is only necessary to take into account that an activity list λ can easily be converted into a pair (λ', d), where λ' is obtained from λ by replacing each activity i by the pair i, i and d is the vector of activity durations. And that the crossover operator has to handle vector λ as well as vector d as has been explained above.

To prove the usefulness of the justification in the 1_PRCPSP we will compare the performance of 1_Random and 1_GA with that of 1_DJRandom and 1_DJGA, respectively. The limitation of 5000 schedules means that, for example, DJRandom calculates1666 random activity lists and applies the DJ to each of the solutions obtained. The same happens with DJGA and their adaptations to 1_PRCPSP.

6. Computational results

In this section we present the results of the computational studies concerning the algorithms introduced in the Section 3. The experiments have been performed on a personal computer AMD at 400 MHz. The algorithms have been coded in *C*.

As test instances, we have used the standard sets j30 and j120 for the RCPSP. They were generated using PROGEN (Kolisch et al., 1995) under a full factorial experimental design with the following three independent problem parameters: network complexity (NC), resource factor (RF), and resource strength (RS). NC is the average number of non-redundant arcs per activity. RF reflects the average portion of resources requested per activity. RS express the relationship between the resource demand of the activities and the resource availability. The set j120 (j30) consists of 600 (480) projects with 120 (30) non-dummy activities. Both sets require four resource types. Further details of these problem instances are given in Kolisch et al. (1995). They are available in the Project Scheduling Project Library (PSPLIB: http://129.187.106.231/psplib) along with their optimum or best-known makespans. Among the four RCPSP instance sets in PSPLIB, we have selected the j30 set because it is the only set for which the optimal solution is known for all the instances in the set. j120 is the set with the largest and most difficult projects in the library. To use these sets of instances for the 1_PRCPSP it is only necessary to add the possibility of interrupting each activity at most once.

6.1. Global results with a limit of 5000 schedules

Table 1 summarises the results of the experiments in j120 for algorithms Random, 1_Random, DJGA, and 1_DJGA. The first column indicates the type of algorithm the results shown in each row refers to. The second and third columns contain the average percentage deviation from the critical path makespan (**CP deviation**), which is a lower bound both for the RCPSP and 1_PRCPSP. The second (third) column refers to the CP deviations calculated for the original (adapted) RCPSP (1_PRCPSP) algorithm. The fourth column indicates the percentage of instances in which an algorithm for the 1_PRCPSP has obtained a better solution than the corresponding RCPSP algorithm. We will call this the **percentage of improved instances**.

Let RCPSP(i) and 1_PRCPSP(i) be the solution obtained in the instance i by a certain type of algorithm in the RCPSP and in the 1_PRCPSP, respectively. We define the improvement obtained by this type of algorithm in the instance i as (RCPSP(i) − 1_PRCPSP(i))/1_PRCPSP(i). Column five (six) shows the **average (maximum) improvement** obtained by each type of algorithm over all instances in j120.

Table 2 shows the same information as Table 1, but regarding the set j30. The second (third) column refers to the deviations with respect to the optimal solutions of the RCPSP. There are instances for which the solution found in the 1_RCPSP is better than the optimal one in the RCPSP. In these cases we have a negative deviation.

It can be observed that there is an important difference in CP deviation between the algorithms with and without interruption, with better results when pre-emption is allowed, especially in j120. In this set, there is a difference in CP deviation of 3.77% between the random solutions in the 1_PRCPSP and in the RCPSP. There is a percentage of improved instances of more than 80% and an average improvement of 2.40%. In j30, the difference in deviation from the optimal solution is 0.75% with a percentage of improved instances of approximately 30% and an average improvement of 0.76%. As we have mentioned before, this average improvement gives information about the difference in average quality of the solutions in both problems. So, the data indicates that the search space in the 1_PRCPSP is of better quality than that of the RCPSP – as might have been expected.

In j120 (j30), there is also an important difference in CP deviation (from the optimal solution) of 2.88% (1.24%) between DJGA and 1_DJGA. There is a percentage of improved instances of approximately 70% (40%) and an average improvement of approximately 2% (1.3%). We have already mentioned in the previous section that DJGA is currently one of the best heuristic algorithms for the RCPSP, so these figures are very meaningful. The best result in j120 with a limit of 5000 schedules in this problem is currently held by HGA (Valls et al., 2002). It obtains a CP deviation of 32.54%, which means a 2.19% of difference with 1_DJGA.

These tables do not offer information about the instances where the algorithms for the RCPSP have obtained a better solution than the corresponding 1_PRCPSP algorithms. In j120 (j30), and for the Random type of algorithm, the percentage of instances where this happens is 2% (4.17%), whereas the minimum improvement percentage (in contrast to the maximum improvement percentage) is − 1.33% (− 2.94%). None of the solutions obtained by DJGA if pre-emption is allowed is worse than the one calculated when it is not permitted.

Summing up, we can conclude that the interruption clearly favours the achievement of better solutions within the same limit on the number of schedules generated. Besides, the adaptation from one algorithm in the RCPSP to one in the 1_PRCPSP is extremely easy, once the 1_serial

Table 1
Computational results on the set j120

Type of algorithm	RCPSP	1_PRCPSP	% imp. inst.	aver. imp.	max. imp.
Random	47.51	43.74	83.50	2.40	6.70
DJGA	33.24	30.35	71.50	1.94	8.26

Table 2
Computational results on the set j30

Type of algorithm	RCPSP	1_PRCPSP	% imp. inst.	aver. imp.	max. imp.
Random	0.95	0.20	32.29	0.76	7.84
DJGA	0.20	−1.04%	40	1.28	8.70

SGS has been developed (which also relies very heavily on the serial SGS). Therefore, beginning from scratch it is approximately as easy – or as difficult – to program heuristic algorithms for the RCPSP as for the 1_PRCPSP. In addition, it is very easy to transform most existing RCPSP procedures into 1_PRCPSP ones. So, if a real problem admits pre-emption, it would be sensible to program a heuristic algorithm that takes advantage of it.

It is worth noting that the deviation of DJGA from the optimal solutions for the RCPSP in j30 is negative, −1.04%. This means that the gap between the optimal solutions for the RCPSP and the 1_PRCPSP is at least 1%.

The impact of the pre-emption is not the same in the two sets. The differences in average improvement, percentage of improved instances, and maximum improvement are bigger in j120 than in j30, especially for the Random algorithm. The data concerning the differences in j30 are much more in concordance with the results obtained by Demeulemeester and Herroelen in the Patterson set, principally the percentage of improved instances in the Random algorithm. Here, it is 32.29% and there, working with optimal algorithms, it was 30 out of 110, which means 27.27%. However, the percentage increases to 40% when a better quality algorithm is used, and the maximum improvement in both algorithms is bigger than the one noted in the Patterson set; 4 and 6 units (7.84% and 8.70%) versus 1 unit in the Patterson set.

6.2. The influence of problem characteristics

In Table 3 (4), the impact of RS, RF, and NC on the percentage of improved instances and on the average improvement relative to algorithm types Random and DJGA in instance set j120 (j30) is examined (Table 4).

RS has a strong impact on both measures of improvement. The lower the resource strength, the higher the improvement. At any rate, it is clear that pre-emption is especially useful when instances are most difficult to solve. This happens when RS is low, both when heuristic and optimal algorithms are used (cf. Hartmann, 1999). However, it also gives better results with easier instances, except when there are no restrictions in the resources (RS = 1). In this case, the ES schedule (every activity beginning at its earliest start time) optimally solves problems RCPSP and 1_PRCPSP and therefore no improvement can be obtained by allowing interruption. If we disregard the instances for which RS = 1, the percentage of improved instances in j30 for the algorithm Random (DJGA) increases to 43.06% (53.33%), and the average improvement to 1.01% (1.71%). Notice also that the settings for the parameter RS are different in both instance sets, being smaller in j120 than in j30. So, one can say that the j120 instances are in general more difficult to solve than the j30 instances.

The behaviour of the difference in relation to NC is just the contrary in j120 and j30. In j120, the average difference increases as NC grows whereas in j30

Table 3
j120 results classified according to problem characteristics

		RS					RF				NC		
		0.1	0.2	0.3	0.4	0.5	0.25	0.5	0.75	1	1.5	1.8	2.1
DJGA	% imp. inst.	99.17	95.00	78.33	61.67	23.33	51.33	72.00	83.33	79.33	66.50	70.00	78.00
	aver. imp.	3.74	2.62	1.86	1.15	0.34	1.50	2.31	2.30	1.66	1.81	1.91	2.11
Random	% imp. inst.	98.33	96.67	85.00	80.00	57.50	58.67	90.00	93.33	92.00	77.50	83.50	89.50
	aver. imp.	3.13	3.23	2.45	1.97	1.22	1.45	2.84	2.96	2.34	2.17	2.39	2.63

Table 4
j30 results classified according to problem characteristics

		RS				RF				NC		
		0.2	0.5	0.7	1	0.25	0.5	0.75	1	1.5	1.8	2.1
DJGA	% imp. inst.	73.33	58.33	28.33	0.00	23.33	46.67	43.33	46.67	42.50	41.88	35.63
	imp. perc.	2.46	1.78	0.89	0	0.89	1.68	1.28	1.29	1.53	1.30	1.02
Random	% imp. inst.	59.17	47.50	22.50	0.00	22.50	36.67	31.67	38.33	33.13	33.75	30.00
	imp. perc.	1.19	1.18	0.68	0.00	0.84	0.89	0.65	0.67	0.93	0.67	0.68

it decreases as NC grows. The effect of RF is not monotonously increasing or decreasing.

6.3. The influence of the limit on the number of schedules generated

It has been proved that pre-emption permits obtaining better solutions if there is a limit of 5,000 schedules. It could be possible that, if more schedules were computed, this gap would disappear. In order to evaluate this possibility we have designed the following experiment. We have run DJGA in j120 with a limit of 10,000, 25,000 and 100,000 schedules, both in the RCPSP and 1_PRCPSP. Each algorithm is allowed to have a different number of individuals in the population in each problem. The results are shown in Table 5. The first column indicates the limit on the number of schedules, while the rest of columns contain the same information as in Tables 1 and 2.

As we can see in columns 2 and 3, both algorithms are capable of calculating better solutions if more schedules are evaluated. Hence, the adaptation of DJGA to the pre-emption problem has resulted in a good quality algorithm for 5000 schedules that produces better results when more schedules are allowed. The difference in CP deviation between the algorithms in both problems is bigger than 2% in the four cases, although it decreases from nearly 3% in 5000 and 10,000 schedules to 2.4% in 100,000. There is also a decrease from nearly 2% of average improvement in 5000 and 10,000 schedules to 1.6% in 100,000. So, the differences when pre-emption is allowed decreases (in this algorithm) when more schedules are calculated, but there still exists an important gap with 100,000 schedules. This is also supported by the fact that the percentage of improved instances does not decrease much, it is still around 70% in 100,000 schedules, and that its maximum improvement percentage is the biggest of the four. Taking everything into account, the pre-emption seems to be also useful when many schedules are calculated.

Regarding the j30 set the average percentage deviation of the 1_DJGA solutions – with a limit of 100,000 schedules – from the optimal solutions of the RCPSP is 1.22%. This figure increases to 1.63% when the trivial instances (RS = 1) are disregarded and to 1.97% when only the instances with RS = 0.2 and RS = 0.5 are considered. This means that the average percentage deviation of the optimal solutions of the 1_PRCPSP from the optimal solutions of the RCPSP is at least 1.22%. Notice that these figures are lower bounds for the case in which the optimal solutions of the 1_PRCPSP – and therefore of the PRCPSP – are considered. It is also interesting to note that, regarding the Patterson instance set, the average percentage deviation of the optimal solutions of the PRCPSP from the optimal solutions of the RCPSP is 0.879%.

6.4. The impact of justification

We are now going to study the impact of justification when pre-emption is permitted. If $A(i)$ denotes the solution obtained by a certain algorithm A in the instance i, the percentage of improvement caused by DJ in this instance is (A(i)-DJA(i))/DJA(i).

Table 6 shows the average percentage of improvement caused by DJ in the Random algorithm (second row) and in GA (third row). Note that these two algorithms, together with their versions with DJ, calculate 5,000 schedules at most. The first column of the table indicates the pair of algorithms (one algorithm with and without DJ) compared to in the rest of the columns. The second (fourth) column contains the results in the RCPSP in the set j30 (j120). The third (fifth) column contains the same information, but refers to the 1_PRCPSP.

All algorithms in both problems and instance sets perform better with DJ than without it. We already know that the DJ greatly improves the RCPSP

Table 5
j120 results for DJGA with increasing number of schedules

Limit of schedules	RCPSP	1_PRCPSP	% imp. inst.	aver. imp.	max. imp.
5,000	33.24	30.35	71.50	1.94	8.26
10,000	32.73	29.78	71.33	1.99	8.26
25,000	31.98	29.34	70	1.81	7.74
100,000	31.10	28.71	68.67	1.63	8.67

Table 6
Impact of justification in the RCPSP and 1-PRCPSP

	RCPSP j30	1_PRCPSP j30	RCPSP j120	1_PRCPSP j120
Random versus DJRandom	0.61	1.11	6.53	6.91
GA versus DJGA	0.02	0.49	2.61	3.79

algorithms in the set j120. This improvement is not so great in j30, up to the point that GA and DJGA obtain approximately the same results. Concerning problem 1_PRCPSP, DJ improves both algorithms in both sets, and the improvement is greater than in the RCPSP. So, it seems clear that the conclusions concerning the justification reached for the RCPSP (Valls et al., 2005) are also valid for the pre-emption case:

1. The justification can be easily added to a wide range of algorithms, since the pre-emption of activities can be easy handled when justifying.
2. Its incorporation greatly improves the quality of the solutions obtained, without having to calculate more schedules. In fact, in the 1_PRCPSP the improvement has been bigger than in the RCPSP.

It is obvious that the first statement is also true for every kind of discrete pre-emption. There is no reason to believe that the second statement – at least, the first part – is not also true for them.

7. Concluding remarks

We have shown how three basic elements of many heuristic algorithms for the RCPSP can be adapted to deal with the pre-emptive case when a maximum of one interruption per activity is allowed. These adaptations can easily be extended to more general pre-emptive problems. We have also proposed a strategy to adapt many RCPSP heuristics to the pre-emptive case without significantly changing the procedure – apart from the above three adaptations.

The results of the computational experiments performed with four RCPSP heuristic algorithms on the standard j30 and j120 sets indicate that the introduction of pre-emption does significantly help in decreasing the project length with respect to that of the no interruption case. We consider this as the main conclusion of this line of work as it contradicts the validity of previous conclusions and motivates research to further investigate the effects of interruption on project length – not only in the RCPSP, but also in more general project scheduling problems. Notice that the computational results on the set j30 support the same conclusion regarding exact procedures.

The good performance of 1_DJGA proves the effectiveness of the procedures 1_serial SGS and the corresponding justification. Both can be easily adapted, together with the codification, to other types of discrete pre-emption, e.g. when only a subset of activities can be interrupted, each a pre-determined low number of times, such as 1, 2 or 3. Besides, the fitness of 1_DJGA can be readily modified to incorporate penalisations for each interruption, thus being able to (heuristically) solve the case when the cost of interrupting an activity is not negligible.

The other important fact we have shown in this paper is that justification is a very simple, easy to implement, and profitable technique when pre-emption is allowed. We have been able to show the great increase in quality that some algorithms show when justification is added – maintaining the same number of schedules calculated.

Acknowledgements

This research was partially supported by the Ministerio de Ciencia y Tecnología under contract TIC2002-02510 and the Agencia Valenciana de Ciencia y Tecnología, GRUPOS03/174.

References

Ballestín, F. (2002). Nuevos métodos de resolución del problema de secuenciación de proyectos con recursos limitados. Unpublished PhD Dissertation, Universidad de Valencia.

Blazewicz, J., Lenstra, J.K., Rinooy Kan, A.H.G., 1983. Scheduling subject to resource constraints: Classification and complexity. Discrete Applied Mathematics 5, 11–24.

Bock, D.B., Patterson, J.H., 1990. A comparison of due date setting, resource assignment and job preemption heuristics for the multiproject scheduling problem. Decision Sciences 21, 387–402.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E., 1999. Resource-constrained project scheduling: notation, classification, models, and methods. European Journal of Operational Research 11, 3–41.

Crandall, K.C., 1973. Project planning with precedence lead/lag factors. Project Management Quarterly 4 (3), 18–27.

Demeulemeester, E., Herroelen, W., 1992. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. Management Science 38, 1803–1818.

Demeulemeester, E., Herroelen, W., 1996. An efficient optimal procedure for the preemptive resource-constrained project scheduling problem. European Journal of Operational Research 90, 334–348.

Demeulemeester, E., Herroelen, W., 2002. Project scheduling – A research handbookInternational Series in Operations Research and Management Science, vol. 49. Kluwer Academic Publishers.

Hartmann, S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. Naval Research Logistics 45, 733–750.

Hartmann, S., 1999. Project scheduling under limited resources: Models, methods, and applicationsLecture Notes in Economics and Mathematical Systems, vol. 478. Springer, Berlin, Germany.

Hartmann, S., Kolisch, R., 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. European Journal of Operational Research 127, 394–407.

Herroelen, W., Demeulemeester, E., De Reyck, B., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25, 279–302.

Icmeli, O., Erenguc, S.S., Zappe, C.J., 1993. Project scheduling problems: A survey. International Journal of Operations & Production Management 13 (11), 80–91.

Kaplan, L.A. (1988). Resource-constrained project scheduling with preemption of jobs. Unpublished PhD Dissertation, University of Michigan.

Kaplan, L.A. (1991). Resource-constrained project scheduling with setup times. Unpublished paper, Department of Management, University of Tenessee, Knoxville.

Kolisch, R. (1995) Project Scheduling under resource constraints – Efficient heuristics for several problem classes, Phisica, Heidelberg.

Kolisch, R., Sprecher, A., Drexl, A., 1995. Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41, 1693–1703.

Kolisch, R., Padman, R., 2000. An integrated survey of deterministic project scheduling. Omega 29, 249–272.

Lino, P. (1997). Planificación de Proyectos en Diagramas de Precedencias. Unpublished PhD Dissertation, Universidad de Valencia.

Neumann, K., Schwindt, C., Zimmermann, J., 2003. Project Scheduling with Time Windows and Scarce Resources. Springer.

Özdamar, L., Ulusoy, G., 1995. A survey on the resource-constrained project scheduling problem. AIIE Transactions 27, 574–586.

Patterson, J.H., 1984. A comparison of exact procedures for solving the multiple constrained resource project scheduling problem. Management Science 30 (7), 854–867.

Slowinski, R., 1980. Two approaches to problems of resource allocation among project activities – A comparative study. Journal of the Operational Research Society 31 (8), 711–723.

Sprecher, A., Kolisch, R., Drexl, A., 1995. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. European Journal of Operational Research 80, 94–102.

Tsubakitani, S., Deckro, R.F., 1990. A heuristic for multiproject scheduling with limited resources in the housing industry. European Journal of Operational Research 49, 80–91.

Valls, V., Ballestín, F., Quintanilla, S. (2002). An hybrid genetic algorithm for the RCPSP. Technical Report, Departamento de Estadística e Investigación Operativa, Universidad de Valencia.

Valls, V., Ballestín, F., Quintanilla, S., 2005. Justification and RCPSP: A technique that pays. European Journal of Operational Research 165, 375–386.

Weglarz, J. (Ed.), 1999. Project Scheduling. Recent Models, Algorithms and Applications. Kluwer Academic Publishers.

Weglarz, J., 1981. Project scheduling with continuously-divisible, doubly constrained resources. Management Science 27 (9), 1040–1053.

Wiest, J.D., 1964. Some properties of schedules for large projects with limited resources. Operations Research 12, 395–418.