Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 165 (2005) 375–386
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Justification and RCPSP: A technique that pays ☆

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

[a] *Dpto. de Estadística e Investigación Operativa, Facultad de Matemáticas, Universitat de Valencia, Dr. Moliner, 50, 46100 Burjassot, Valencia, Spain*

[b] *Dpto. de Economía Financiera y Matemática, Facultad de Económicas y Empresariales, Universitat de Valencia, Avda. de los Naranjos, s/n, Edificio Departamental Oriental, Valencia, Spain*

Received 1 December 2002; accepted 1 May 2003
Available online 8 June 2004

Abstract

The objective of this paper is to show that justification is a simple technique that can be easily incorporated in diverse algorithms for the resource-constrained project scheduling problem—improving the quality of the schedules generated without generally requiring more computing time. The results of incorporating this technique in 22 different algorithms are shown. Fifteen of the new algorithms that use double justification outperform seven of the best heuristic algorithms that do not use justification. The tests have been performed on the standard test set j120 for the RCPSP generated using ProGen.

Keywords: Resource-constrained project scheduling; Justification; Heuristic algorithms

1. Introduction

The classical resource-constrained project scheduling problem (RCPSP) continues to be an active area of research attracting in recent years increasing interest from researchers and practitioners in the search for better solution procedures. We refer to the surveys provided by Icmeli et al. (1993), Özdamar and Ulusoy (1995), Herroelen et al. (1998), Brucker et al. (1999) and Kolisch and Padman (2001). For solving the RCPSP exactly, the most competitive algorithms seem to be those of Demeulemeester and Herroelen (1992), Brucker et al. (1998), Mingozzi et al. (1998), Klein and Scholl (1999), Sprecher (2000) and Dorndorf et al. (2000). Since the RCPSP is NP-hard in the strong sense (Blazewicz and Lenstra, 1983), it is not surprising that only small-sized problem instances with up to 60 activities can be solved exactly in a satisfactory manner. Therefore, heuristic solution procedures remain as the only feasible method of handling practical resource-constrained project scheduling problems.

Many heuristic approaches have been proposed for RCPSP (we refer to the aforementioned

☆ This research was partially supported by the CICYT under contract TAP99-1123 and by the Generalitat Valenciana under contract FP 198-CB-12-303.
* Corresponding author.
E-mail addresses: vicente.valls@uv.es (V. Valls), francisco.ballestin@uv.es (F. Ballestín), maria.quintanilla@uv.es (S. Quintanilla).

0377-2217/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2004.04.008

surveys). Kolisch and Hartmann (1999) have given an overview of heuristics focusing on X-pass methods (single pass methods, multi-pass methods, sampling procedures) and metaheuristics (simulated annealing, genetic algorithms, Tabu search). Apart from the algorithms covered by this computational comparison, different heuristics have been proposed by Li and Willis (1992), Özdamar and Ulusoy (1996), Merkle et al. (2000), Klein (2000), Nonobe and Ibaraki (2001), Alcaraz and Maroto (2001), Tormos and Lova (2001), Hartmann (2002), Möhring et al. (2003), and Valls et al. (2003, 2004). Few of these researchers have paid due attention to a simple and efficient technique for improving the quality of schedules—the justification of schedules.

In an early paper, Wiest (1964) introduced the concepts of left-justified and right-justified schedules and a procedure for associating a right-justified schedule to a given left-justified schedule by local left-shifting of all activities. The objective being to extend the unambiguously defined concepts of activity slack and critical path in the unlimited resource case to the case where resources are limited. Since then few researchers have used these, or related concepts, when designing algorithms for the RCPSP. Although both procedures differ in important aspects, Wiest's procedure can be considered as an antecedent of the justification technique to be defined later this paper.

Li and Willis (1992) propose an iterative forward/backward scheduling technique for project scheduling under general resource-constraints. Basically, the procedure iteratively applies serial forward and backward scheduling until no further improvement in the project duration can be obtained. The activity finish (start) times of a forward (backward) schedule determine the activity priorities for the next backward (forward) schedule. In other words, they perform left and right-justification of schedules as defined in this paper.

The heuristic technique of Tormos and Lova (2001) is an iterative method where each iteration is a two-stage procedure. The first stage builds a feasible schedule by means of a regret-based biased random sampling procedure. In the second stage, this schedule is right-justified and then the resulting schedule is left-justified.

Valls et al. (2003, 2004) propose two different metaheuristic algorithms for solving the RCPSP. They are based on different approaches. The first one is a non-standard implementation of fundamental concepts of Tabu Search without explicitly using memory structures, whereas the second one is a population-based procedure. However, both incorporate the techniques of right and left-justification inside the algorithmic scheme for two reasons. Firstly, to improve the quality of solutions obtained by other methods for a marginal computational cost. Secondly, to alternatively aim the search at two regions with different characteristics (strategic oscillation).

In this paper, we wish to highlight the first reason. We intend to show that justification is a simple technique that can be incorporated easily into various algorithms and produces notable improvements in the quality of the schedules with a small, and generally favourable, change in computing time.

The remainder of the paper is organized as follows. Section 2 presents a formal definition of the RCPSP along with some definitions. Section 3 describes the set of computational experiments designed to show the benefits of using double justification whereas the results obtained are analysed in Sections 4–6. Finally, a summary and some concluding remarks appear in Section 7.

2. Problem description and definitions

The RCPSP may be stated as follows. A project consists of a set of activities $V = \{1, \ldots, n\}$ where each activity has to be processed without preemption. The dummy activities 1 and n represent the beginning and the end of the project. The duration of an activity j is denoted by d_j where $d_1 = d_n = 0$. 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 $r_{1k} = r_{nk} = 0$, $k = 1, \ldots, K$. All parameters are assumed to be non-negative integer valued. There are precedence relations of the finish-start type with a zero parameter value (i.e., $FS = 0$) defined between the activities. $S_j(P_j)$

is the set of successors (predecessors) of activity j. It is assumed that $1 \in P_j,\ j = 2,\ldots,n$, and $n \in S_j,\ j = 1,\ldots,n-1$. The objective of the RCPSP is to find a schedule S of the activities, i.e., a set of starting times $(s_1, s_2, \ldots, s_n)$ where $s_1 = 0$ and the precedence and resource-constraints are satisfied, in such a way that the schedule length $T(S) = s_n$ is minimised.

An **active** (or **left active**) **schedule** can be defined as a schedule where no activity can be started earlier without delaying some other activity—or violating the constraints. A formal definition can be found in Sprecher et al. (1995). Analogously, a **right active schedule** is a schedule where no activity can be finished later without advancing some other activity, or violating the constraints, or increasing the makespan. Given a schedule S, to **justify an activity $j \neq n(1)$ to the right (left)** consists of obtaining a schedule S' such that $s'_i = s_i,\ i \neq j$, $s'_j \geqslant s_j$ $(s'_j \leqslant s_j)$ and s'_j is as large (small) as possible. Given a schedule S, the right (left) justification of the activities j in decreasing (increasing) order of $f_j = s_j + d_j(s_j)$ provides a right active (left active) schedule $\boldsymbol{S^R}(\boldsymbol{S^L})$ that is called the **right (left) justification of $\boldsymbol{S}$**. $S^R(S^L)$ is not unique, and depends on the tie-breaking rule used. In this paper, we assume that ties are randomly broken; except in the case of the Hartmann algorithm, where the tie break rule operates by activity number. If $s^R_1 > 0$ $(s^L_n < T)$, then S^R (S^L) is shorter than S. It is not difficult to see that $T(S^R) \leqslant T(S)$ $(T(S^L) \leqslant T(S))$. Notice that $(S^R)^L$ is a left active schedule. We can say that $\boldsymbol{DJ}(\boldsymbol{S}) = (S^R)^L$ is the result of **double justifying** the schedule S.

It is worth noting that Wiest's procedure is not a schedule justification procedure as it generally does not justify the activities in the required order. For the same reason, iterative backward/forward scheduling (Özdamar and Ulusoy, 1996) is generally not a double justification procedure. Also, the bi-directional scheduling schemes of Klein (2000) do not justify all schedules but only those activities scheduled in a backward direction.

A **schedule generation scheme (SGS)** schedules the activities of a project in a stage by stage manner until a feasible schedule is attained. Two different SGS are usually considered, the serial SGS and the parallel SGS. In each stage, the **serial SGS** (Kelley, 1963) selects an eligible activity according to some priority rule, and schedules it at the earliest precedence and resource-feasible start time. An activity is called **eligible** if all its predecessors are already scheduled. On the contrary, the stages in the **parallel SGS** are defined by time periods in chronological order. In each stage (period) t, the activities that can be precedence and resource-feasibly started at t, are ordered according to some priority rule. This order is then used for scheduling the activities at t, if resources permit. A formal description of both SGS can be found in Kolich (1995).

3. Experiment design

As it is not possible to analyse the benefits of incorporating the techniques of justification in all types of algorithms, we have designed a set of experimental computations using three sets of very diverse algorithms.

In the first set we have added simple, very fast, and well-known algorithms; including two single pass methods and two multi-pass methods which were ranked in first positions in a previous computational study (Kolich, 1995). One of the single pass methods employs the serial SGS with the minimum latest finish time (LFT) priority rule. The other employs the parallel SGS with the worst-case slack (WCS) priority rule. One of the multi-pass methods uses regret-based biased random sampling under the LFT rule with $\alpha = 1$ (Drexl, 1991). The other simply consists in randomly generating priority value vectors, applying the serial SGS and choosing the best schedule obtained.

The second set includes more evolved algorithms that, nevertheless, cannot be considered state-of-the-art algorithms. We specifically designed 16 simple population-based algorithms for these tests. All the algorithms share the same procedure for generating the initial population; and basically the same evolutionary scheme reduced to the same minimal expression. In each iteration, a given evolutionary scheme acts on the schedules of the actual population and generates a

new population. These evolutionary schemes are constructed from binary operators that combine characteristics of the two schedules to generate a new schedule. These 16 algorithms of this second set are obtained by combining in every possible way the defined evolutionary schemes (4) and the operators (4). The diversity of the operators and evolutionary schemes creates a variety of algorithms. We have also included in this set a very simple simulated annealing algorithm.

Finally, the third set consists of a Hartmann algorithm (Hartmann, 1998)—one of the best state-of-the-art algorithms that does not use justification.

The idea of alternatively justifying the schedule to the right and to the left can be implemented in many different ways. All the algorithms we are going to describe work with active schedules, that is to say, with left active schedules. Therefore, it seems that the simplest way to incorporate justification is to justify each schedule generated by the algorithm twice consecutively first to the right and then to the left. Therefore, we can refer to algorithms and algorithms with double justification (DJalgorithm).

The experiment consists of comparing the quality of the schedules obtained for each of the 22 algorithms before, and after, incorporating the double justification. To provide a basis for the comparison, we limited the total number of generated schedules to 5000 where appropriate. This limit has been used in previous computational experiments. The experiments were performed on a personal computer with AMD processor at 400 MHz. The algorithms were coded in *C*.

As test instances, we used the standard set j120 for the RCPSP generated using ProGen (Kolisch et al., 1995). The set j120 consists of 600 projects with four resource types and 120 activities. These instances were generated under a full factorial experimental design with the following three independent problem parameters: network complexity, resource factor, and resource strength. Details of these problem instances are given in Kolisch et al. (1995) and Kolisch and Sprecher (1997). They are available in the project scheduling problem library (PSPLIB) along with the optimum, or best known, values that have been obtained by various authors over the years. As of November 2002, the optimum solution for j120 instances had only been discovered in approximately 34% of the instances. This shows the difficulty represented by the j120 set for current algorithms.

4. First set of experiments

In the first set of experiments we have included four algorithms with, and without, double justification.

The first (second) algorithm applies the serial (parallel) SGS with the LFT (WCS) rule for each instance. If the generated solution is subsequently justified twice then an algorithm with double justification is obtained.

The third algorithm randomly generates 5000 active schedules for each instance and saves the best one. The same algorithm, but with double justification, randomly generates 1666 schedules and these are then doubly justified. In this way, some $1666 \times 3 = 4998$ schedules are generated in total.

The fourth algorithm is the regret-based biased random sampling with the LFT rule and with $\alpha = 1$. It is applied 5000 times to each instance. The double justification version generates 1666 schedules in the same way and then double justifies them.

Table 1 summarises the results of our first set of experiments. The first column indicates the algorithm referred to the results shown in each row. Each cell in the second column contains two numbers. The first number consists of the average percentage deviation from the critical path makespan lower bound—which is obtained by computing the length of a critical path in the resource

Table 1
Computational results for the first set of algorithms on j120

Algorithm	Original/DJoriginal
Serial LFT	48.11/**43.34**
Parallel WCS	43.58/**39.42**
Complete Random	47.51/**38.36**
Regret Sampling LFT	42.02/**37.47**

relaxation of the problem. The second number in bold presents the same information but referring to the algorithms with double justification.

We can add some statistics. The double justification has improved 77.17% of the schedules obtained by the serial LFT algorithm and 90% of those generated by the parallel WCS. Of the $1666 \times 600 = 999{,}600$ randomly generated schedules, only 170 cannot be improved by the double justification. In other words, the double justification has improved 99.38% of the randomly generated schedules. This percentage is an estimation of the probability that double justification will improve a randomly chosen active schedule. Obviously, this percentage could vary if we reduce the sample space to regions of high quality active schedules. Some 95.36% of the schedules generated by the regret sampling LFT algorithm were improved by double justification. It is also worth mentioning that in no instance the original algorithms produced better solutions than their DJ versions.

Table 1 shows that the double justification greatly improves the quality of the solutions generated for all the algorithms. The improvement in the average percentage deviation from the critical path makespan is 9.15 for the complete random algorithm; while for the other algorithms the figure is between 4.16 and 4.77. The difference can be explained by the fact that the quality of the randomly generated schedules is very unequal. This often produces low quality random schedules, and as a result these can be improved much more than higher quality schedules. It is worth noting that the serial LFT and parallel WCS algorithms generate a schedule for every instance, while the other two algorithms generate 5000.

In their independently developed work, Tormos and Lova (2001) obtain similar results. They find that the quality of the solutions obtained by the serial and parallel SGS with the LFT rule and by the regret-based biased random sampling method with the LFT rule and both SGS improves when DJ (or BF as they call it) is applied. They perform their experiment on the standard set j30 for the RCPSP generated using ProGen (Kolisch et al., 1995), which consists of 480 projects with four resource types and 30 activities.

It is also worth noting that the double justification does not increase the computing time of the complete random and regret sampling LFT algorithms.

5. Second set of experiments

In this section we intend to extend the experiments to include more complex algorithms that, nevertheless, cannot be considered state-of-the-art algorithms for the RCPSP. To achieve this we have designed 16 new population-based algorithms and a simulated annealing algorithm specifically for this experiment.

5.1. Population-based algorithms

All population-based algorithms share the same procedure for generating an initial population, and basically, the same iterative evolutionary scheme reduced to the minimal expression. In each iteration, the current population is renewed by considering new generated schedules.

To construct an initial population we have used the same procedure as in Valls et al. (2003). In short, the initial population consists of the schedules constructed from both nine well-known priority rules and ten randomly generated priority value vectors. Both schedule generating schemes (serial and parallel) are applied where possible.

There are four quite similar evolutionary schemes that differ on the procedure applied to generate new schedules and on the population renewal policy. The generating procedures are based on operators that combine the characteristics of two given schedules to generate a sequence of schedules $C_1, C_2, \ldots, C_q$.

Below, we describe the operators that can be used in each evolutionary scheme. To define the operators, we make use of the priority value representation of Lee and Kim (1996). A priority value vector is a vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ that assigns a real-valued number γ_i to each activity i. The serial SGS can be used to obtain an active schedule $S(\gamma)$ from a vector γ of priority values by selecting at each stage the eligible activity j with the lowest priority γ_j, ties are broken by activity

number. We assume that the lower the priority value then the more important is the activity.

A schedule can be represented by an infinite number of priority value vectors. Nevertheless, given an active schedule S we define γ_S as the single vector that assigns the integer numbers between 1 and n to the activities in growing order of start times and ties broken by an activity number. If S is active it satisfies $S(\gamma_S) = S$.

5.1.1. Sum operator

The sum operator is an operator that given two schedules A and B constructs the sequence of schedules $\{S(\gamma^1),\ldots,S(\gamma^q)\}$, where $\gamma^j = \left(1-\frac{j}{q+1}\right)\gamma_A + \frac{j}{q+1}\gamma_B$, $j = 1,\ldots,q$. The sequence of schedules $\{S(\gamma^1),\ldots,S(\gamma^q)\}$ is in the 'segment' that joins the schedules A and B and the greater is j then the more it resembles A. For a more detailed study of the sum operator see Valls et al. (2004).

5.1.2. Change operator

The change operator is an operator that given two schedules A and B constructs the sequence of schedules $\{S(\gamma^1),\ldots,S(\gamma^q)\}$, where γ^j, $j = 1,\ldots,q$, is constructed by randomly changing $\left\lfloor\frac{n}{q+1}\right\rfloor$ different unchanged components of γ^{j-1} for the same components of the vector γ_B and where $\gamma^0 = \gamma_A$.

5.1.3. Percentage operator

The percentage operator is an operator that given two schedules A and B constructs the sequence of schedules $\{C_1, C_2,\ldots, C_q\}$ where each schedule C_j, $j = 1,\ldots,q$, is constructed by applying the serial SGS in the following way. At each stage of the serial SGS, the set of eligible activities is calculated and a random number (p) is generated between 0 and 1. If $p \leqslant \frac{q-j+1}{q+1}$, then an eligible activity is selected with the lowest priority value according to γ_A; otherwise, an eligible activity is selected with the lowest priority value according to γ_B. Therefore, $100\frac{q-j+1}{q+1}\%$ of the times we choose an activity according to γ_A and the remaining times according to γ_B.

5.1.4. Window operator

The window operator is an operator that given two schedules A and B constructs the sequence of schedules $\{C_1, C_2,\ldots, C_q\}$ where each schedule C_j, $j = 1,\ldots,q$, is constructed by applying the serial SGS in the following way. At each stage, the eligible activities are sorted according to γ_A. If the number of eligible activities is greater than $j+1$ then the first $j+1$ are selected, otherwise, all are selected. In either case, a selected activity is sequenced with the lowest priority value according to γ_B. Therefore, schedule A determines the activities that are the most promising for sequencing in the current stage; while schedule B selects the most adequate activity from among them.

Notice that all operators obtain a sequence of schedules that begin resembling A, but progressively less resemble A, and more resemble B.

Below, we briefly describe for each evolutionary scheme the contents of an iteration. *POPsize*, q and *nsol* are input parameters to the procedures.

5.1.5. Evolutionary scheme 1

To each pair of schedules (A, B) of the current population, apply an operator to generate the schedules $\{C_1, C_2,\ldots, C_q\}$. The new population consists of the *POPsize* generated schedules with best makespan.

5.1.6. Evolutionary scheme 2

To each pair of schedules (A, B) of the current population, apply an operator and select the first schedule generated C_1. The new population consists of the *POPsize* selected schedules with best makespan.

5.1.7. Evolutionary scheme 3

For each schedule A of the current population, randomly generate a schedule B. Apply an operator to the pair (A, B) and select the first schedule generated C_1. The new population consists of the *POPsize* selected schedules with best makespan.

5.1.8. Evolutionary scheme 4

For each schedule A of the current population, randomly generate *nsol* schedules B_j, $j = 1,\ldots,$ *nsol*. To each pair of schedules (A, B_j), apply an operator and select the first schedule generated C_1. The schedule A is replaced for the best selected schedule if this improves the makespan of A.

Notice that we have introduced diversity into the evolutionary schemes by three means. Firstly, by generating either q schedules or just one, between two given schedules. Secondly, by either combining pairs of schedules that both belong to the current population—or pairs of schedules where one schedule belongs to the current population and the other has been randomly generated. Finally, by changing the renewal policy in the fourth evolutionary scheme.

The 16 population-based algorithms are obtained by combining in every possible way the evolutionary schemes (4), and the defined operators (4). The diversity of operators and schemes induces a variety of algorithms.

5.2. Computational results for the population-based algorithms

The objective of this section is to analyse the effect of including the double justification in relatively simple algorithms that produce relatively good solutions but cannot be considered state-of-the-art algorithms. We have not spent much effort optimising the selection of the parameter values. We have arbitrarily fixed the value, or the range, of the possible parameter values. Firstly, we have fixed the value of the parameter q for various reasons. The value of q is the same for all the algorithms that use the same operator. The sum operator was used in Valls et al. (2004). In order to harmonise with the work in this paper, we have fixed $q = 15$ for the group of eight algorithms (four without and four with DJ) that use the sum operator. An operator similar to the window operator was used in Valls et al. (2003) and after referring to the experimentation in this paper we have fixed $q = 8$ for the group of eight algorithms that use the window operator. For the eight algorithms that use the change operator we have fixed $q = 11$. In this way, each of the schedules in the sequence $C_1, C_2, \ldots, C_q$ differs from the earlier schedule by 10 changes. Finally, we have fixed $q = 9$ for the eight algorithms that use the operator percentage. In this way, the 'similarity percentage' of each schedule C_j, $j = 1, \ldots, q$, with the schedule A is reduced 10% each time.

The set of possible values of the *POPsize* parameter is limited to $\{5, 10, 15, 20, 25, 30\}$. The parameter *nsol* is only used in scheme 4, and its value equals 1 in the algorithms that use the sum and window operators. It varies within the set $\{5, 10, 15, 20\}$ in the algorithms that use the other two operators.

Using some preliminary tests we fixed for each of the 32 algorithms the 'best' combination of parameter values for *POPsize* and *nsol*. Once the previous values of the parameters have been set, the number of iterations is calculated so that the total of generated schedules is equal to, or less than, 5000. The selection of the values of the parameters determines the number of schedules generated by each algorithm. This number varies between 4400 and 5000 with an average value of 4766.56. Table 2 shows the values selected by each of the 32 algorithms. The rows indicate the operators used and the columns show the schemes. The boxes showing schemes 1, 2, and 3 contain two numbers. The first (second) shows the value of the *POPsize* for the version without (with) justification of the resulting algorithm for the version without (with) justification of the algorithm resulting from combining the operator shown in the row and the scheme shown in the column. The figures corresponding to scheme 4 also show the value of *nsol*.

Table 3 summarises the percentage deviations from the lower bound critical path for instance set

Table 2
Values of selected *POPsize* and *nsol*

Op/scheme	Scheme 1	Scheme 2	Scheme 3	Scheme 4
Sum	15/5	30/20	20/5	15, 1/5, 1
Changes	15/5	25/15	15/10	5, 15/5, 15
Percentage	15/5	30/15	20/10	5, 5/5, 5
Window	20/10	30/15	20/10	5, 1/5, 1

Table 3
Computational results for the population-based algorithms on j120

Op/scheme	Scheme 1	Scheme 2	Scheme 3	Scheme 4
Sum	38.96/**35.18**	39.78/**35.46**	38.78/**34.35**	39.30/**35.37**
Changes	38.05/**34.50**	37.80/**34.02**	38.71/**34.92**	38.44/**34.59**
Percentage	38.12/**34.60**	38.16/**34.27**	38.60/**34.78**	38.04/**34.13**
Window	38.87/**34.46**	39.11/**34.08**	39.27/**35.49**	39.30/**35.31**

Table 4
Instance-wise comparison

Op/scheme	Scheme 1	Scheme 2	Scheme 3	Scheme 4
Sum	6	0	7	2
Changes	13	0	2	1
Percentage	10	0	3	6
Window	0	1	1	1

j120. The first column indicates the operator used by the algorithmic schemes mentioned in the first row. Each cell in the table refers to an algorithm formed by the combination of the scheme shown in the column and the operator shown in the row. Each cell contains two numbers, one in a normal font and the other in bold. The normal font figure refers to an algorithm without double justification, and the bold figure refers to the algorithm with double justification.

We can see that all the algorithms have been improved by applying the double justification. Before justifying, the percentage deviations from the lower bound critical path varied from a minimum value of 37.80% to a maximum value of 39.78%. After the double justification, this range was reduced to $[34.02, 35.49]$. The improvement of the percentage deviations from the lower bound critical path varied from a minimum of 3.52 to a maximum of 5.09.

It is worth noting that in no case was the average computing time of an algorithm with double justification longer than the same algorithm without justification.

Table 4 presents the number of instances for which the original algorithms produced better solutions than their DJ counterpart.

It is interesting to compare these results with those obtained by the state-of-the-art algorithms that do not use justification. Table 5 shows the percentage deviations from the lower bound critical path obtained by various authors on j120 with an upper limit of 5000 generated schedules except the algorithm of Möhring et al. that generated 3675 schedules on average. We can see that all the unjustified algorithms in Table 3 generate worse quality solutions than all the algorithms in Table 5. Nevertheless, when the double justification is introduced, 13 of the 16 algorithms produced better solutions than all the algorithms in Table 5 and the remaining three outperformed all but two algorithms.

Table 5
Computational results of state-of-the-art algorithms without justification on j120

Algorithm type	Author(s)	CP_dev
Self-adapting GA	Hartmann, 2002	35.35
Ant colony optimisation	Merkle et al., 2000	35.43
Lagrangian Heuristic	Möhring et al., 2003	36.2
Genetic algorithm	Alcaraz and Maroto, 2001	36.57
Activity list GA	Hartmann, 1998	36.74
Truncated B&B[a]	Dorndorf et al., 2000	37.1

[a] With run time limit of 300 seconds.

Möhring et al. (2003) showed that if a local search algorithm is added to the Lagrangean heuristic then the deviation from the critical path makespan lower bound is reduced to 35.3%. This result is not shown in Table 5 because the limit of 5000 is apparently exceeded in the new version of

the algorithm. Also, the result (35.62%) obtained by Tormos and Lova (2001) is not included in Table 5 because their algorithm applies double justification. As limiting the number of schedules generated and evaluated is meaningless with the algorithms in Valls et al. (2003, 2004) we have also not included their results. These results were 34.53 and 31.58 percentage deviations from the lower bound critical path, respectively.

5.3. A simulated annealing algorithm

The 16 algorithms studied in the preceding sub-section are population-based algorithms that generate new schedules by combining two existing schedules. In this sub-section, we will analyse the effect of applying double justification to a completely different algorithm: a simulated annealing algorithm (SA).

SA is a straightforward implementation of a basic simulated annealing algorithmic scheme. The temperature decreases linearly, the number of schedules visited at each temperature is constant and a neighbour of a schedule is obtained by applying the β-biased random sampling method, β-BRM, introduced by Valls et al. (2003). For more details, we refer to Ballestín (2001).

Table 6 shows the computational results obtained. Row SA refers to algorithm SA when limiting the number of schedules generated to 5000. Row DJSA refers to the algorithm obtained from SA when 1666 schedules are generated by β-BRM and then double justified. The average computation time in seconds is shown in column Av_CPU.

We can see that algorithm SA has been improved by applying double justification. Before justifying, the percentage deviation from the lower bound critical path was within the range of those of the population-based algorithms (Table 3)—and worse than those obtained by state-of-the-art algorithms (Table 5). After justifying, CP_dev improves by 4.44% reaching a solution quality significantly superior than that of the state-of-the-art algorithms in Table 5. SA obtained better solutions than DJSA in only three instances. It is worth noting that this improvement in solution quality is simultaneous with a decrease in the CPU time. The reason being that the operation of justifying a schedule to the left (right) can be implemented by applying the serial SGS with the activities ordered in increasing (decreasing) order of their start (finish) times on the original (reverse) project network (Ballestín, 2001). Thus it is that DJ runs faster than any biased sampling method. Its computational simplicity is another advantage of DJ.

Table 6
Computational results for algorithm SA with, and without, justification

	CP_dev	Av_CPU
SA	38.03	3.16
DJSA	33.59	2.21

6. Third set of experiments

In this section we will analyse the effect of applying double justification to an algorithm that offers quality solutions. For this task we have selected the Hartmann's genetic algorithm, one of the best state-of-the-art algorithms that does not use justification.

In this test we use a genetic programming activity list algorithm (Hartmann, 1998) that we have implemented (GA). Firstly, we ran the program with a limit of 5000 schedules, then with a limit of 10,000 schedules, and then with 5000 schedules after incorporating double justification. The size of the population in the unjustified algorithms were 100 and 200, respectively while the justified algorithm used 50. The results appear in Table 7 below.

We can see that in the case of the upper limit of 5000 generated schedules, the double justification

Table 7
Computational results for GA algorithm with, and without, justification

	CP_dev (%)	Av_CPU
GA 5000	37.00	1.55
GA 10,000	36.18	3.08
DJGA 5000	33.24	1.60

improves the quality of the solutions by 3.76% although with an incremental margin of 0.05 seconds in computing time. To discover if the increase in quality could be caused by the increase in computing time we ran the GA without justification and a limit of 10,000 schedules. The results obtained dismiss this hypothesis. DJGA 5000 outperforms GA 10,000 both in terms of quality solution and CPU time needed.

It is important to emphasise that while the Hartmann genetic activity list algorithm occupies fifth place in Table 5, the new DJGA 5000, which is obtained from the former algorithm by simply adding double justification, notably outperforms all the state-of-the-art algorithms in Table 5. We can also mention that GA 5000 obtained better solutions than DJGA 5000 in only two instances.

To complete the information about the rest of the state-of-the-art algorithms referred to in Table 5, we have summarised the information provided by the authors in their papers regarding CPU times and computers used: self-adapting GA of Hartmann (14.05 seconds on a Pentium-based computer at 133 MHz); Merkle et al. (25 seconds on a Pentium III, 500 MHz); Möhring et al. (65 seconds on a Sun Ultra 2 workstation, 200 MHz, 1 Gbyte memory); Alcaraz and Maroto (20 seconds on a PC166); activity list GA of Hartmann (14.15 seconds on a Pentium-based computer at 133 MHz) and Dorndorf et al. (205 seconds on a PC200).

7. Conclusions

In this paper we have intended to show that justification is a simple technique that can be easily incorporated in many algorithms for the RCPSP—and that it produces notable improvements in schedule quality without generally increasing computing time.

As it is not possible to analyse the benefits of incorporating justification to all types of algorithms, we have designed a set of experimental computations on three very different types of algorithm. In the first type of algorithm we included well-known, fast, and simple algorithms. We have seen how a specific form of justification, double justification, greatly improves the quality of the solutions generated by all of the considered algorithms. It is worth emphasising that these two priority rules, which have been so often quoted in the literature over the years, have been easily improved by simply adding double justification.

In the second set of algorithms we included more complex algorithms that offer better quality solutions but cannot be considered state-of-the-art for the RCPSP. We designed 16 new population-based algorithms specifically for this experiment. Two additional principles guided the design of these algorithms—simplicity and diversity. For simplicity's sake, the evolutionary scheme of all the algorithms is reduced to the minimal expression. Diversity is obtained using four distinct binary operators and four distinct algorithmic schemes for generating schedules. In this case too, the double justification notably improved the quality of the solutions generated by all the algorithms considered. It is worth emphasising that before justification, all the designed algorithms generated worse quality solutions than all of the state-of-the-art algorithm cited in Table 5. Nevertheless, after applying double justification, 13 of the 16 algorithms generated better quality solutions than all the state-of-the-art algorithms in Table 5—and the remaining three outperformed all but two of the algorithms. These improvements were obtained without any additional computational cost. We have also included in this second set a completely different but still medium quality algorithm—a simple simulated annealing algorithm. The significant improvement in solution quality obtained indicates that the benefits of applying double justification are not restricted to population-based algorithms. It is also worth noting that this improvement was obtained with a reduction of 30% in CPU time.

Finally, we analysed the effects of applying double justification to an algorithm that offers quality solutions. To achieve this, we selected an activity list GA from Hartmann (1998). This is one of the best state-of-the-art algorithms that does not use double justification. The computational results obtained indicate that, also in this case, double justification significantly improves the quality of the solutions generated. In fact, DJGA

clearly outperforms all the state-of-the-art algorithms appearing in Table 5.

Concluding, we incorporated double justification in 22 diverse heuristic algorithms that generate solutions of varying quality—and the result was always a notable improvement in solution quality. In our opinion, these results show that double justification should always be considered when designing an algorithm for RCPSP. This is especially true as its implementation is, in general, a small part of the work necessary to code an algorithm for RCPSP.

References

Alcaraz, J., Maroto, C., 2001. A robust genetic algorithm for resource allocation in project scheduling. Annals of Operations Research 102, 83–109.

Ballestín, F., 2001. Nuevos métodos de resolución del problema de secuenciación de proyectos con recursos limitados, Ph.D. Thesis, Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain.

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.

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 112, 3–41.

Brucker, P., Knust, S., Schoo, A., Thiele, O., 1998. A branch and bound algorithm for the resource-constrained project scheduling problem. European Journal of Operational Research 107, 272–288.

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

Dorndorf, U., Pesch, E., Phan-Huy, T., 2000. A branch-and-bound algorithm for the resource-constrained project scheduling problem. Mathematical Methods of Operations Research 52, 413–439.

Drexl, A., 1991. Scheduling of project networks by job assignment. Management Science 37, 1590–1602.

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

Hartmann, S., 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistic 49, 433–448.

Herroelen, W., De Reyck, B., Demeulemeester, E., 1998. Resource-constrained project scheduling: A survey of recent developments. Computers and Operations Research 25 (4), 279–302.

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

Kelley Jr., J.E., 1963. The critical-path method: Resources planning and scheduling. In: Muth, J.F., Thompson, G.L. (Eds.), Industrial Scheduling. Prentice-Hall, Englewood Cliffs, NJ, pp. 347–365.

Klein, R., 2000. Scheduling of resource-constrained projects. Kluwer Academic Publishers, Boston, MA.

Klein, R., Scholl, A., 1999. Scattered branch and bound: An adaptive search strategy applied to resource-constrained project scheduling. Central European Journal of Operations Research 7, 177–201.

Kolich, R., 1995. Project scheduling under resource constraints—efficient heuristics for several problem classes, Phisica-Verlag, 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., Hartmann, S., 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In: Weglarz, J. (Ed.), Project Scheduling. Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Boston, MA, pp. 147–178.

Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 29, 249–272.

Kolisch, R., Sprecher, A., 1997. PSPLIB—A project scheduling library, European Journal of Operational Research 96, 205–216. (downloadable from http://www.bwl.uni-kiel.de/Prod/psplib/index.html).

Lee, J.K., Kim, Y.D., 1996. Search heuristics for resource constrained project scheduling. Journal of the Operational Research Society 47, 678–689.

Li, R.K.-Y., Willis, J., 1992. An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56, 370–379.

Merkle, D., Middendorf, M., Schmeck, H., 2000. Ant colony optimization for resource-constrained project scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference, Las Vegas, NV, 2000, pp. 893–900.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1998. An exact algorithm for project scheduling with resource constraints based on a new mathematical formulation. Management Science 44, 714–729.

Möhring, R.H., Schulz, A.S., Stork, F., Uetz, M., 2003. Solving project scheduling problems by minimum cut computations. Management Science 49 (3), 330–350.

Nonobe, K., Ibaraki, T., 2001. Formulation and Tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, pp. 557–588.

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

Özdamar, L., Ulusoy, G., 1996. A note on an iterative forward/backward scheduling technique with reference to a procedure by Li and Willis. European Journal of Operational Research 89, 400–407.

Sprecher, A., 2000. Solving the RCPSP efficiently at modest memory requirements. Management Science 46 (5), 710–723.

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.

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling. Annals of Operations Research 102, 65–81.

Valls, V., Quintanilla, S., Ballestín, F., 2003. Resource-constrained project scheduling: A critical activity reordering heuristic. European Journal of Operational Research 149 (2), 282–301.

Valls, V., Ballestín, F., Quintanilla, S., 2004. A population-based approach to the resource-constrained project scheduling problem, Annals of Operations Research, in press.

Wiest, J.D., 1964. Some properties of schedules for large projects with limited resources. Operations Research 12 (3), 395–418.