Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 149 (2003) 268–281
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version

K. Bouleimen *, H. Lecocq

Service de Robotique et Automatisation, Institut de Mécanique et de Génie civil Bât. B52/3 Chemin des chevreuils, 1 Sart Tilman 4000 Liège, Belgique

Abstract

This paper describes new simulated annealing (SA) algorithms for the resource-constrained project scheduling problem (RCPSP) and its multiple mode version (MRCPSP). The objective function considered is minimisation of the makespan. The conventional SA search scheme is replaced by a new design that takes into account the specificity of the solution space of project scheduling problems. For RCPSP, the search was based on an alternated activity and time incrementing process, and all parameters were set after preliminary statistical experiments done on test instances. For MRCPSP, we introduced an original approach using two embedded search loops alternating activity and mode neighbourhood exploration. The performance evaluation done on the benchmark instances available in the literature proved the efficiency of both adaptations that are currently among the most competitive algorithms for these problems.

Keywords: Project scheduling; Resource constraints; Simulated annealing

1. Introduction

This study deals with the resource-constrained project scheduling problem (RCPSP) and its extension to the multiple mode case (MRCPSP). The paper proposes a metaheuristic approach based on simulated annealing (SA) to solve these problems when considered for minimisation of total project duration. Both RCPSP and MRCPSP are popular problems that are both widely treated and regularly surveyed in the literature. The reader is referred to the papers of Herroelen et al. [17,18], Brucker et al. [8] and Kolisch and Padman [25] for an introduction and overview of different formulations of project scheduling problems.

A large number of contributions for solving the RCPSP are available in the literature. The surveys [18,25] provide detailed descriptions of the characteristics, representations and classification schemes. The methods suggested can be classified into three categories: exact methods [9–11,28,29],

* Corresponding author. Tel.: +32-4-366-94-55; fax: +32-4-366-95-05.
E-mail address: k.bouleimen@student.ulg.ac.be (K. Bouleimen).

0377-2217/03/$ - see front matter
doi:10.1016/S0377-2217(02)00761-0

priority rules [4,12,20,22,27] and metaheuristic approaches such as tabu search [3,31], genetic algorithms [2,14,19] or SA [6]. The RCPSP is known to be $\mathcal{NP}$-hard and the best exact methods are able to optimise small sized projects usually with less than 60 activities. For larger and more complex instances, heuristics and metaheuristic methods constitute a real challenge to other solution techniques for providing the best trade-off between performance, extensibility and ease of implementation. The recent benchmark results [2,16,24] show that the genetic algorithms proposed by Alcaraz and Maroto [2] and Hartman [14], and our SA are the best performing methods currently available for RCPSP. The first part of this paper provides the detailed description of the new SA adaptation.

For the MRCPSP we also find several studies published in the literature where some exact methods [15,30,32,33], priority rules [5,13,23] and metaheuristic approaches [6,14] are proposed. As a generalisation of RSPSP, this problem is also $\mathcal{NP}$-hard. The most effective exact procedures are not able to optimally solve some benchmark instances with 30 activities.

The metaheuristic adaptations for this problem that are still being investigated and improved seem the most promising. The SA procedure for the RCPSP introduced previously has been extended to the MRCPSP, and is able to support the use of renewable and non-renewable resources. This new procedure was tested on benchmark test problems using an original search strategy and proved to be very competitive. The second part of this paper concerns the description of this original adaptation.

The paper is organised as follows. After the introduction and problems description (Section 2), Section 3 presents the SA method. Section 4 describes the new adaptation of SA to the considered project scheduling problems and presents the general framework of the proposed procedures. In Section 5 we explain the design of the parameters of the procedures through preliminary experiments, then detail in Section 6 the results obtained in the performance evaluation. The last section presents general conclusions and comments.

2. Problems description and formulations

2.1. RCPSP

An RCPSP consists of a set $\mathcal{A} = \{1, \ldots, J\}$ of activities that have to be scheduled under resource and precedence constraints. Precedence constraints force any activity j not to be started if any one of its immediate predecessors in the set $\mathcal{P}red_j$ has not been completed. The constraints defined on a set $\mathcal{R} = \{1, \ldots, R\}$ of renewable resources are the limitations of the available quantities $\mathcal{Q}_r$ per period of time for every resource r. Each activity j is processed in a non-pre-emptive way, and uses an amount q_{jr} units of the resource r during the period of its duration d_j.

Scheduling a project consists in assigning start dates to all its activities such that all precedence and resource constraints are fulfilled. A precedence and resource feasible schedule determines a makespan value that is the total project duration.

For the RCPSP formulation, the objective function considered is the minimisation of the makespan. This problem is denoted as $m, 1/cpm/C_{\max}$ following the classification proposed by Herroelen et al. [17] and as $PS/prec/C_{\max}$ following that proposed by Brucker et al. [8].

2.2. MRCPSP

In MRCPSP, the precedence constraints are defined on set $\mathcal{A}$ by giving a set of predecessors $\mathcal{P}red_j$ for every activity j. To be processed, the activities usually require three categories of resources: R types of renewable resources from set $\mathcal{R}$ constrained with limited quantities $\mathcal{Q}_r$, non-renewable resources from a set $\mathcal{N} = \{1, \ldots, N\}$ where each resource n is constrained with a limited quantity of $\mathcal{Q}_n$ units available for the entire project duration, and doubly constrained resources from a set $\mathcal{D} = \{1, \ldots, D\}$, for which every resource d is available with a quantity $\mathcal{Q}_d$ limited on both total project duration and time periods. Doubly constrained resources can be considered as parts of the two first resources categories. Each activity j can be processed in one of $\mathcal{M}_j$ possible modes, where each mode m defines a relation

between its resource requirements and its duration. Thus, when processed in a mode m, activity j will have a duration d_{jm} during which it uses q_{jmr} units of the renewable resource r and consumes q_{jmn} units of the non-renewable resource n.

In the MRCPSP, the objective is to find a mode combination and a schedule that minimise the project makespan. Herroelen et al. [17] denote this problem as $m,1T/cpm,mu/C_{\max}$ while in Brucker et al. [8] it is denoted as $MPS/cpm/C_{\max}$.

3. Simulated annealing

SA is a local search method that finds its inspiration in the physical annealing process studied in statistical mechanics [1]. An SA algorithm repeats an iterative neighbour generation procedure and follows search directions that improve the objective function value. While exploring solution space, the SA method offers the possibility to accept worse neighbour solutions in a controlled manner in order to escape from local minima. More precisely, in each iteration, for a current solution x characterised by an objective function value $f(x)$, a neighbour x' is selected from the neighbourhood of x denoted $N(x)$, and defined as the set of all its immediate neighbours. For each move, the objective difference $\Delta = f(x') - f(x)$ is evaluated. For minimisation problems x' replaces x whenever $\Delta \leqslant 0$. Otherwise, x' could also be accepted with a probability $P = e^{(-\Delta)/T}$. The acceptance probability is compared to a number $y_{\text{random}} \in [0,1]$ generated randomly and x' is accepted whenever $P > y_{\text{random}}$.

The factors that influence acceptance probability are the degree of objective function value degradation Δ (smaller degradations induce greater acceptance probabilities) and the parameter T called temperature (higher values of T give higher acceptance probability). The temperature can be controlled by a cooling scheme specifying how it should be progressively reduced to make the procedure more selective as the search progresses to neighbourhoods of good solutions. There exist theoretical schedules guaranteeing asymptotic convergence toward the optimal solution. They require however infinite computing time. In practice, much simpler and finite computing time schedules are preferred even if they do not guarantee an optimal solution.

A typical finite time implementation of SA consists in decreasing the temperature T in S steps, starting from an initial value T_0 and using an attenuation factor α $(0 < \alpha < 1)$. The initial temperature T_0 is supposed to be high enough to allow acceptance of any new neighbour proposed in the first step. In each step s, the procedure generates a fixed number of neighbour solutions N_{sol} and evaluates them using the current temperature value $T_s = \alpha^s T_0$. The whole process is commonly called "cooling chain" or also "markov chain". Adaptation of SA to an optimisation problem consists in defining its specific components: a solution representation of the problem, a method for the objective function value calculation, a neighbour generation mechanism for solution space exploration and a cooling scheme including stopping criteria. These adaptation steps for our new adaptations of SA for both RCPSP and MRCPSP are described below.

4. Simulated annealing for project scheduling

4.1. Introduction

When adapted efficiently to optimisation problems, SA is often characterised by fast convergence and ease of implementation. These characteristics motivate the choice of SA for $\mathcal{NP}$-hard combinatorial optimisation problems.

The first adaptation of SA to RCPSP was done by Boctor [6]. The procedure proposed was able to outperform a tabu search algorithm and some of the best priority rules then available [6]. However, to confirm this performance, further investigations and improvements had to be accomplished on the main questionable points of the study: a specific design of the SA parameters, a validation of the results on larger and more complex instances and a performance evaluation with reference to more competitive methods. And though we opted for a solution representation based on activity lists previously used by Boctor et al. [24], the new adaptation, characterised by an original cooling

scheme, considerably improved the efficiency of the procedure as shown in Section 6.

In the same paper Boctor also proposed an extension of the SA to multiple mode problems. However, the formulation he considered does not include the use of non-renewable resources. Hence, we preferred to extend our procedure to a more general model and suggest a new design for neighbourhood exploration.

4.2. Solution procedure for RCPSP

4.2.1. Solution representation and schedule generation scheme

The solution representation is an important component of SA. It has to be designed such as to allow easy neighbour generation and fast calculation of Δ. It must also guarantee accessibility for the entire solution space. Various types of solution representations for the RCPSP have been proposed in the literature and are reviewed in the paper of Kolisch and Hartmann [24]. In this study we use an activity list representation where a precedence-feasible solution is represented by an ordered list of activities. Each activity in the list appears in a position after all its predecessors. Fig. 1A represents an example of a feasible activity list for a project with 60 non-dummy activities.

As a scheduling procedure, we use a serial schedule generation scheme (SGS) [24] that is adapted to the activity list representation. This SGS alternates two operations of "start time assignment to activities" and "time incrementing" until all activities of the project, including the dummy finish activity, are scheduled. The procedure is initialised by fixing the time at $S_{\text{time}} = 0$ after scheduling and finishing the dummy start activity. The "start time assignment" operations are always performed when the time is fixed, and consist in consecutively testing the unscheduled activities one after the other following the list order. However, no activity can be tested before its list-predecessor has not been scheduled. The "time incrementing" operations are necessary each time the procedure is unable to schedule the current activity tested.

For example, when at a fixed time $S_{\text{time}} = t$, an activity j is tested for scheduling, the procedure first checks all precedence constraints on j. Then if all activities of the set $\mathscr{P}red_j$ are finished, it checks if the available amounts of resources are sufficient to schedule it. If all constraints can be fulfilled, j is

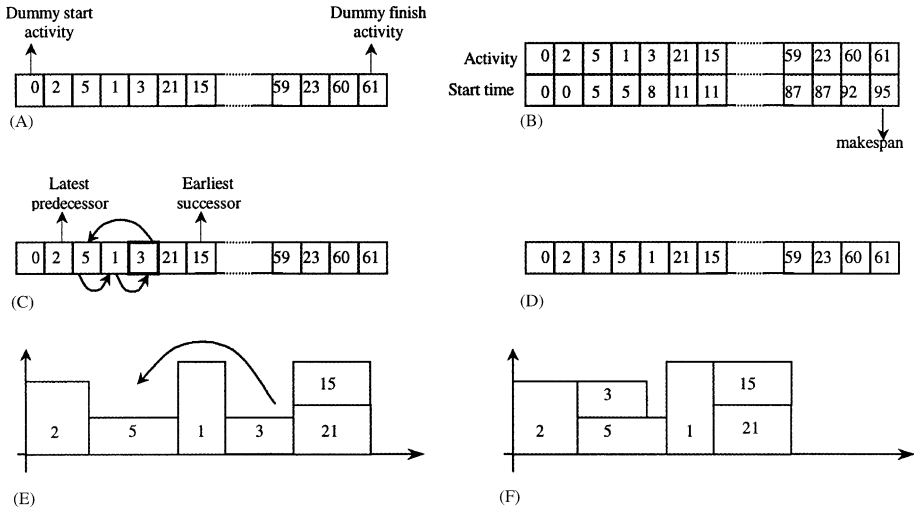

Fig. 1. Solution representation and neighbour generation for RCPSP. (A) Example of a feasible solution representation, (B) example of a feasible schedule, (C) activity 3 and its new position are selected randomly, (D) feasible neighbour solution after the cyclical shift, (E) semi-active partial schedule [34], (F) global shift of activity 3.

scheduled at time t and, before testing the next activity, the quantities q_{jr} required by j are retrieved from the resource amounts currently available. However if scheduling j would induce any constraint violation, the "start time assignment" operation is interrupted although some list-successors of j may be schedulable at that time. A "time incrementing" operation is then performed until the earliest time where some finished activities recover the feasible conditions to schedule activity j. These operations are repeatedly alternated until all project activities are scheduled. The project makespan is the start time of the dummy finish activity. The complete schedule is the association of the priority list and the start time list (Fig. 1B). We note that, as proven in [21], the serial SGS generates active schedules only. Our SGS generates also semi-active schedules [34] because it includes the list-precedence constraint while searching for the earliest precedence and resource-feasible time to schedule any activity.

4.2.2. Neighbour generation

The neighbour generation is performed as follows: on the current feasible list we randomly select an activity and calculate the positions of its latest predecessor and earliest successor (Fig. 1C). The selected activity can be moved anywhere within these two positions to respect the precedence constraints. The new position is also randomly chosen. When a move is possible, the new list is obtained by a cyclical shift of all the activities placed between the old and the new positions (Fig. 1D).

The cyclical shift operator used in our procedure is very important for makespan improvement when applied on semi-active schedules [34] (Fig. 1E). In several cases, the cyclical shift on the list results in a global shift of the moved activity on the schedule (Fig. 1F). Global shifts of activities are essential for makespan improvements on semi-active schedules and keep optimal schedules reachable.

4.2.3. Cooling scheme

This section describes a new cooling scheme proposed for this SA adaptation. Our objective is to improve the efficiency of the procedure by using a search strategy that is adapted to the RCPSP by introducing some problem-dependent parameters. We first present the main characteristics of this new schedule in order to introduce the general framework of the procedure. However, all the modifications were made according to observations and results obtained in the preliminary experiments described in Section 5. The cooling scheme is structured with the following components:

- The search is performed with a multiple cooling chain (C) and restarted each time from a different initial solution. The solutions tested in every step s of the cooling chain are progressively increased using the expression $N_s = N_{s-1}\,(1 + h.s)$. In this way the procedure is allowed to extend its search while it progresses to neighbourhoods of good solutions. Parameter h is used to modulate the step length.
- The initial degradation Δ is limited to a proportion of $\sim$20% of $f(x_0)$ in the first step (this is done by deducing the initial temperature value $T_{0\max}$ from the expression $P(\Delta_{\max}) = 0.01$ that assigns a very low acceptance probability to $\Delta_{\max}$). On the other hand, for large projects the procedure can be accelerated by a partial rescheduling mechanism.
- In the last step, the procedure is forced to leave a solution unchanged after N_{average} neighbours generated (N_{average} is an average neighbourhood size). At the end of the procedure, the neighbourhood of the best solution found is totally explored.

The finite time implementation of SA implies the use of stopping criteria. It could be the total process time, the number of neighbours generated or any condition on the objective value. With our adaptation of SA to RCPSP, the process time necessary to generate a fixed number of solutions is almost proportional to the project size J. So, the main stopping criterion used is the total number of visited solutions N_{tot}. However, the procedure is stopped whenever it meets the critical path value corresponding to the unconstrained problem (calculated in the first initialisation step).

4.2.4. General framework of the procedure

Fig. 2 presents the general framework of the new SA procedure for the RCPSP. It suggests performing the search using multiple cooling chains, where each chain is designed to allow enough time for deep exploration of the path followed by the procedure. Each chain is composed of several steps where the number of visited solutions is progressively increased while the temperature decreases. For every instance, the initial solution x_0 is obtained by the shortest process time (SPT) heuristic. The initial temperature $T_{0\max}$ is then calculated using the maximal degradation allowed ($\Delta_{\max}$) and the initial objective function value $f(x_0)$.

4.3. Solution for MRCPSP

4.3.1. Solution representation and schedule generation scheme

For MRCPSP two lists are used to represent a feasible solution: a precedence-feasible activity list associated to a resource-feasible mode list. The feasibility of mode lists consists in guaranteeing that the total amount of every non-renewable resource n required during the scheduling process does not exceed the available amounts $\mathcal{Q}_n$.

The list association is used to generate a project schedule. First it is necessary to fix all mode-dependent requirements of the project activities using the mode list. Then the same SGS used for the RCPSP is performed to assign start dates to all activities and consequently determine the makespan. Three lists represent a complete schedule. The makespan value of a MRCPSP depends on both activities and mode lists. So it is possible to generate a neighbour solution modifying any one of the lists or all of them.

4.3.2. Neighbour generation

In this study we suggest using "activity neighbourhood" and "mode neighbourhood" as separate exploration techniques in a two-stage procedure. The first stage consists in searching for an objective function value improvement using only mode neighbourhood exploration. The second stage is launched each time a better value is found, and consists in an activity neighbourhood exploration performed on the RCPSP sub-problem corresponding to the new solution obtained in the

Step1: ***Read*** *the project data and Calculate its CP value*
Step2: ***Calculate*** x_0 *and* $f(x_0)$ *using the SPT heuristic*
Step3: ***Store*** x_0 *as best schedule* x_{best} *and* $f(x_0)$ *as best objective function value* f_{best}
Step4: ***Store*** *the activity list of* x_0 *as current list and* $f(x_0)$ *as* $f_{current}$
Step5: ***Read*** *the SA parameters:* N_0, h, T_{0max}, α, S *and* C
Step6: ***For*** C *Chains* ***Do***
    $T=T_{0max}$
    $N_s=N_0$
    For S *steps* ***Do***
        $N_s=N_s(1+h\,.\,s)$
        For N_p *neighbours* ***Do***
            Generate *a neighbour* x' *of the current solution* $x_{current}$
            Calculate $f(x')$
            Calculate $\Delta=f(x')-f(x)$
            If $\Delta<0$ ***then store*** $x'=s_{current}$ *and* $f(x')=f_{current}$
                If $f(x')<f_{best}$ ***then store*** $x'=x_{best}$ *and* $f(x')=f_{best}$
                If $f_{best}=CP$ *value* ***then exit*** *the procedure*
            Else if $P=e^{(-\Delta)/T}>x_{random}$ ***then store*** x' *and* $f(x')$ *as currents*
        EndDo N
        Calculate $T=\alpha T$
    EndDo S
EndDo C
Step7: ***Explore*** *the Neighbourhood of the best solution*

Fig. 2. SA procedure for RCPSP.

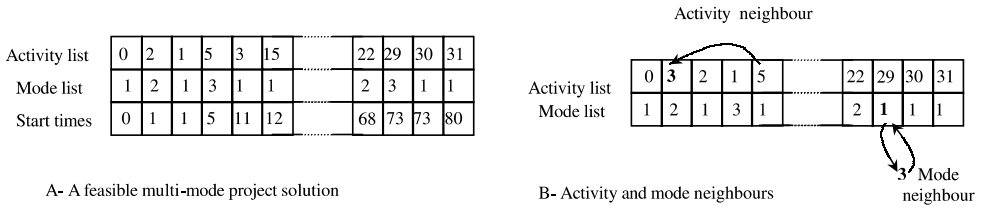

Fig. 3. Solution representation and neighbour generation for MRCPSP.

first stage. The sub-problem is generated by fixing the mode-dependent requirements with the mode list and by removing the non-renewable resources from the resource list.

In order to generate a mode neighbour, the priority list remains unchanged; for a random activity, the current mode is replaced by another one. The new mode is accepted if it respects the non-renewable resource constraints. When only the current mode is feasible, another activity will be selected. The first stage is started from an initial solution that is generated randomly. The new parameters introduced in the MRCPSP are the number of initial solutions (N_{ini}) and the number of mode neighbours that we try to find (N_{mod}).

In the original design we have applied the SA concept in both stages of the procedure. However, after the first computation tests this idea revealed to be inefficient and complex. Thus, the first stage does not use probabilistic acceptance criteria, but only accepts solutions with $\Delta \leqslant 0$. However, the mode-neighbour search is bounded by a limited number of trials for makespan improvement (N_{bound}), after which the stage is restarted with a new initial solution.

For MRCPSP, the search for feasible mode neighbours induces variable numbers of trials from one instance to another, and consequently variable process time. The main stopping criterion used was based on a fixed total process time for every instance (Fig. 3).

4.3.3. Procedure for MRCPSP

Fig. 4 shows the framework of the procedure for MRCPSP. We can see that in a first stage (step 4), it performs a search for N_{mod} mode neighbours, starting from several initial solutions N_{ini}. The number of mode neighbours $N_{m\text{tot}}$ tested in the first stage is at maximum equal to $N_{\text{ini}}N_{\text{mod}}$. For each one of the mode neighbours, N_{tot} schedules will be

```
Step1: Read the project data and calculate its CP value
Step2: Read the SA parameters for RCPSP : N0, h, T0max, α, S and C
Step3: Read the additional parameters for multiple mode problems : Nini, Nmod
Step4: For Nini initial solution Do
           Select the modes with the minimum duration values
           Search a feasible mode neighbour x' of the current list x
           For Nmod mode neighbours Do
               Evaluate f(x') then Δ
               If Δ ≤ 0 then do
                   Store the Objective and the List as Current
                   If f(x')<fbest then store the f(x') and s' as best ones
                   Store the current mode list
                   Remove the non-renewable Resources
                   Explore the Sub-Problem with RCPSP Procedure (Step 6)
                   Associate the resulting best list to the current mode list
               Search a new feasible Mode Neighbour
           End Do Nmod
       End Do Nini
```

Fig. 4. SA procedure for MRCPSP.

generated in the second stage. The final schedule consists of the mode list stored in the first stage and the best activity list generated in the second stage.

5. Preliminary computation experiments

The objective of the preliminary experiments was to define a strategy for parameter tuning. It consisted of three tests: evaluation of the effect of the parameters (T, α, S and N_{sol}), observation of the neighbourhood.s characteristics for various instances and trade-off tuning of conflicting parameters. These tests were done on a sample of 100 instances picked from the benchmark sets used for the performance evaluation [26].

The first test consisted in generating several solutions and building the corresponding sets $N(x)$. For each solution, we measured the spread of objective values around $f(x)$ and the size of each set. For the first measure, we noticed in particular that the initial solutions s_0, obtained by the SPT heuristic typically have more than 80% of solutions in $N(x_0)$ within less than 25% of deviation from $f(x_0)$. This means that the Δ allowed in the first step could be limited without excluding a large number of solutions from acceptance. Through the second measure, we determined an average experimental value of neighbourhood size ($N_{average}$) for instances with the same number of activities J and characterised by the same network complexity factor [26]. $N_{average}$ is used to control the local minima traps in the last step. Thus, whenever the procedure keeps a same solution after testing $N_{average}$ neighbours, it is forced to accept the next solution proposed.

In the second test, parameters T, α, S and N_{sol} were tuned with several values and observed separately. It is interesting to note that T and N_{sol} are the most influential parameters on the procedure and have to be tuned precisely, while α and S could be assigned some conventional values. For example, if α is set to 0.25, five steps are sufficient to reach the selectivity desired in the last step (as specified in the second bullet, Section 4.2.3). If a higher value is chosen for α, more steps will be necessary.

On the other hand, testing a constant number N_{sol} of neighbours in all cooling steps results in a drastic reduction in procedure performance. The solutions visited in each step should rather be increased progressively as the search becomes more selective. So, N_{sol} was replaced by N_s, which is increased in each step following $N_s = N_{s-1}(1 + h.s)$ and the total number of visited solutions is $N_{total} = \sum N_s$. We also noticed in this second test that for a same N_{total}, a search with several cooling chains generally performs better than with one chain. And when doing so, it is also better to restart the new cooling chain from a different initial solution.

The last experiment was a trade-off tuning of parameters C, h and N_{tot}. It was performed on a set of 480 projects with 30 activities, as all optimal makespans of these problems are available in the literature [8]. Although 30 activities is not a prohibitive size, it does permit making observations and analyses that are likely to be appropriately generalised to larger problems. Results are summarised in Tables 1 and 2. The values 5000 and 10,000 for N_{tot} were chosen to allow all combinations of h and C when $S = 5$.

The best performances obtained are represented in bold. As one can see, both C and h improve procedure performance. The value of C must be set at the maximum possible at the condition to allow enough length in each cooling chain for a deep exploration. For this point we suggest using

Table 1
Results with 5000 neighbours

5000 solutions	$C = 1$			$C = 2$			$C = 5$		
	Maximum deviation	Average deviation	Optima found	Maximum deviation	Average deviation	Optima found	Maximum deviation	Average deviation	Optima found
$h = 0$	8.86	0.75	359	**9.36**	0.64	369	10.22	0.67	327
$h = 1$	**6.58**	**0.58**	**375**	12.90	**0.54**	**382**	**7.89**	**0.50**	381
$h = 2$	14.58	0.71	370	10.91	0.64	377	12.90	0.61	**385**

Table 2
Results with 10000 neighbours

10000 solutions	$C = 1$			$C = 2$			$C = 5$		
	Maximum deviation	Average deviation	Optima found	Maximum deviation	Average deviation	Optima found	Maximum deviation	Average deviation	Optima found
$h = 0$	7.89	0.47	392	8.33	0.46	393	7.89	0.49	386
$h = 1$	**7.22**	**0.42**	**403**	**6.58**	**0.37**	**402**	**5.26**	**0.26**	**418**
$h = 2$	7.92	0.45	399	7.89	0.46	396	12.90	0.40	410

$N_0 \geqslant N_{\text{average}}$. The introduction of parameter h also revealed to be very efficient for performance improvement. It is used to modulate the length of the steps in the cooling chain in trade-off with C, and once the test conditions (N_{tot}) are known.

Lastly, we notice that the scheduling process is the most time-consuming operation in the SA procedure. In the case of RCPSP, it is not possible to directly deduce the value of Δ by a simple comparison of the old and new lists. It is necessary to reschedule the project to obtain the new makespan. However, for large projects, and as an acceleration scheme, we suggest to partially reschedule the project, starting only from the smallest changed position in the list. This is possible if a resource state history is stored for the current solution. The use of $T_{0\max}$ in the first step is also done with the purpose of saving process time loss induced by exploration of very bad and non-useful neighbourhoods.

6. Performance evaluation

6.1. Procedure for RCPSP

6.1.1. Standard test

The procedure was coded with Microsoft Visual C++1.52 and run on an IBM compatible PC with a 100 MHz Pentium processor and 32 Mbytes of RAM. It was tested on public benchmark instances available in the literature [5,26]. The condition for the test was to allow a search on $N_{\text{tot}} = 1000J$ neighbours for each instance. This condition guarantees equitable process time allowed for similar instances. Process times induced by the N_{tot} neighbours were 30 seconds for $J = 30$, 60 seconds for $J = 60$ and 100 seconds for $J = 90$.

The results obtained are compared to the best-known solutions available in the literature. These results are published on the project scheduling library (PSLIB) *ftp* site at the following address: ftp://www.wiso.uni-kiel.de/pub/operations-research. For sets where all optimal solutions are known, we evaluate average and maximum deviations and proportion of optima found. Otherwise, for sets with heuristic solutions, we give proportions of better, worse and equal solutions and the corresponding average and maximum deviations (Table 3).

6.1.2. Results analysis

After evaluating the results obtained, the main conclusion reached is that the RCPSP procedure is very efficient. It was able to optimise all instances considered as easy or small sized (Patterson set and set with $J = 30$) with acceptable process times. For larger and more complex problems, the

Table 3
Solution for the RCPSP benchmark instances with 30–90 activities and the Patterson problems

Set	Instances	Optima found	Equal	Better	Worse	Average deviation	Maximum deviation	Average time (seconds)	Maximum time (seconds)
Patterson	110	110	–	–	–	0	0	1.436	4.221
30	480	480	–	–	–	0	0	17.468	28.013
60	480	290	57	98	35	0.467	4.45	44.237	60
90	480	267	59	110	66	0.702	3.44	74.212	100

average deviation of less than 1% obtained can be considered as an excellent performance, especially if we consider that we refer to results generated by several procedures where the computation times were not published. The proposed procedure is simple but powerful and clearly improves the performance obtained by previous adaptations of SA to RCPSP. It is important to notice that the procedure was validated on the same set of instances used for the SA parameters tuning, which may lead to a biased performance analysis. However, the good performance stated in our standard test has been confirmed in more restrictive conditions, in a comparison test [26] with competitive methods available for the RCPSP.

6.1.3. Comparison test by Kolisch and Hartmann

In their recent study, Kolisch and Hartmann [26] proposed to compare the available heuristics for the RCPSP. The test consisted in solving the sets with $J = 30$, 60 and 120 using 1000 and then 5000 different solutions, with no other particular restriction. For our case, the test was performed with unique parameter settings for all the instances. The study does not indicate if the other authors did the same. The results in Tables 4–8 (taken from Kolisch and Hartmann's paper) clearly show the very good performance of our SA procedure, which is constantly ranked among the best performing procedures. These benchmark results have been updated with the new results published in Alcaraz and Maroto [2] where the authors propose a robust GA for the RCPSP.

Deeper analyses of the results presented in [24] were proposed later in a paper by Hartmann and Kolisch [16], where the authors studied the influence of problem parameters on procedure performance. They used the results obtained for the

Table 4
Average deviation from optimal solutions ($J = 30$)

Algorithm	Author	1000	5000
GA	Alcaraz, Maroto	0.33	0.12
SA	**Bouleimen, Lecocq**	**0.38**	**0.23**
GA	Hartmann	0.54	0.25
TS	Baar et al.	0.86	0.44
Sampling-Ad.	Kolisch, Drexl	0.74	0.52
Sampling LFT s	Kolisch	0.83	0.53
Sampling-Ad.	Schrimer, Riesenberg	0.71	0.59
Sampling-WCS	Kolisch	1.40	1.28
Sampling LFT p	Kolisch	1.40	1.29
GA	Leon, Ramamoorthy	2.08	1.59

Table 5
Average deviation from best solutions ($J = 60$)

Algorithm	Author	1000	5000
GA	Alcaraz, Maroto	0.91	0.41
GA	Hartmann	0.99	0.45
SA	**Bouleimen, Lecocq**	**1.17**	**0.49**
Sampling-Ad.	Schrimer, Riesenberg	1.26	0.97
Sampling-Ad.	Kolisch, Drexl	1.60	1.26
TS	Baar et al.	1.79	1.54
Sampling LFT s	Kolisch	1.88	1.55
Sampling LFT p	Kolisch	1.83	1.56
Sampling-WCS	Kolisch	1.88	1.56
GA	Leon, Ramamoorthy	2.48	1.82

Table 6
Average deviation from best solutions ($J = 120$)

Algorithm	Author	1000	5000
GA	Alcaraz, Maroto	2.51	0.69
GA	Hartmann	2.59	0.89
SA	**Bouleimen, Lecocq**	**5.73**	**1.86**
Sampling LFT p	Kolisch	2.92	2.32
Sampling-WCS	Kolisch	2.94	2.34
Sampling-Ad.	Schrimer, Riesenberg	3.28	2.55
Sampling-Ad.	Kolisch, Drexl	3.95	3.33
GA	Leon, Ramamoorthy	5.33	3.76
Sampling LFT s	Kolisch	4.78	4.10

Table 7
Average deviation from CP lower bound ($J = 60$)

Algorithm	Author	1000	5000
GA	Alcaraz, Maroto	12.57	11.86
GA	Hartmann	12.68	11.89
SA	**Bouleimen, Lecocq**	**12.75**	**11.90**
Sampling-Ad.	Schrimer, Riesenberg	13.02	12.62
Sampling-Ad.	Kolisch, Drexl	13.51	13.06
Sampling-WCS	Kolisch	13.66	13.21
Sampling LFT p	Kolisch	13.59	13.23
TS	Baar et al.	13.80	13.48
GA	Leon, Ramamoorthy	14.33	13.49
Sampling LFT s	Kolisch	13.96	13.53

Table 8
Average deviation from CP lower bound ($J = 120$)

Algorithm	Author	1000	5000
GA	Alcaraz, Maroto	39.36	36.57
GA	Hartmann	39.37	36.74
SA	**Bouleimen, Lecocq**	**42.81**	**37.68**
Sampling-LFT	Kolisch	39.60	38.75
Sampling-Ad.	Schrimer, Riesenberg	39.65	38.77
Sampling-Ad.	Kolisch, Drexl	40.08	39.08
Sampling-WCS	Kolisch	41.37	4.45
GA	Leon, Ramamoorthy	42.91	40.69
Sampling LFT s	Kolisch	42.84	41.84
–	–	–	–

benchmark set with $J = 30$ given 5000 iterations. They have performed a multi-variate linear regression analysis, with the average deviation from the optimal solution as a dependent variable, and the three problem parameters (resource strength, resource factor and network complexity [26]) as independent variables. The measure values used for performance evaluation and reported in the paper again confirmed the robustness and superiority of the GA of Hartmann and our SA over the other procedure, with an advantage to the GA procedure.

Table 9
Solution for MRCPSP benchmark instances with 10–20 activities

Set (activities)	Average deviation	Maximum deviation	Percentage of optima found (%)	Average time (seconds)	Maximum time (seconds)
10	0.21	7.8	96.3	19,361	50
12	0.19	6.3	91.2	34,807	60
14	0.92	10.6	82.6	51,141	70
16	1.43	12.9	72.8	69,954	80
18	1.85	11.7	69.4	74,823	90
20	2.10	13.2	66.9	88,434	100

Table 10
Solution for MRCPSP benchmark instances with 30 activities

Set (Activities)	Instances	Equal	Better	Worse	Average deviation	Maximum deviation	Average time (seconds)	Maximum time (seconds)
30	560	490	32	38	2.61	13.7	260,845	300

6.2. Procedure for MRCPSP

6.2.1. Standard test

The multiple mode procedure was written with the same code and tested on the same machine as in the RCPSP case. The main stopping criterion applied in the standard test was a limitation of the total process time allowed for the search. For sets with $J = 10$–20 activities, the total time was fixed to $5J$ seconds. For the set with $J = 30$ activities it was $10J$ seconds (300 seconds). Results obtained in this test are summarised in Tables 9 and 10. Table 9 presents the optimisation rate and the average and maximum deviations from optimal solution for the sets with $J = 10$–20 activities (all the optimal solutions are known for these sets). Table 10 concerns the set with $J = 30$ and give worse, equal and better solutions obtained comparing our results to the best-known solutions.

6.2.2. Results analysis

The procedure for MRCPSP is also satisfactory. It is capable of a high optimisation rate with reasonable process time allowed for the search. Performance measures decrease on average with instance size. But some individual observations show several better performances on the large instance concerning convergence speed (compared to the exact procedure used in reference). Thus, for various instances from the sets with 18 and 20 activities, the procedure was up to 50 times faster to find the optimal schedule than the exact algorithm used in the PSLIB (taking into account the difference between processor speeds).

The lower average deviations obtained with the RCPSP procedure can be justified by the fact that multiple modes problems are more complex and applied experiment conditions relatively more restrictive. The results shown in Table 10 show that the procedure was able to improve 32 solutions for the 30 activity problems. Several solutions are still maintained in the best solution list that is continuously updated on the PSLIB *ftp* site. This proves that the current procedure is very competitive and could be considered as among the best available methods.

7. Conclusions

In this paper, we have presented SA adaptations for RCPSP and its multiple mode version. The procedures were designed using information related to some characteristics of project scheduling problems and new search strategies. Tested on public benchmark problems, both procedures revealed to be highly efficient. They are often capable of optimisation, or otherwise provide high quality approximation in reasonable process times. Moreover, the RCPSP procedure outperformed

most of the alternative methods currently available, as shown in a recent comparative study [18]. The MRCPSP procedure was able to improve 32 best solutions for the unsolved problems with 30 activities. The procedures would be useful and efficient for real life problems, since the tuning could be specifically refined for each project at hand. The process times used here offer a very wide margin compared to times allowed for managers to schedule real life projects. These good results encouraged further adaptations to other interesting versions of PSP such as multi-project scheduling with a multi-objective approach [7], and motivated the development of a web-based decision support system for real life project scheduling problems.

References

[1] E.H.L. Aarts, J.H.M. Korst, Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Computing, Wiley, Chichester, 1989.
[2] J. Alcaraz, C. Maroto, A robust genetic algorithm for resource allocation in project scheduling, Annals of Operations Research 102 (2001) 83–109.
[3] T. Baar, P. Brucker, S. Knust, Tabu search algorithms for resource-constrained project scheduling problems, in: S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimisation, Kluwer, 1997, pp. 1–18.
[4] F.F. Boctor, Some efficient multi-heuristic procedures for the resource-constrained project scheduling problem, European Journal of Operational Research 49 (1) (1990) 3–13.
[5] F.F. Boctor, Heuristics for scheduling projects with resource restrictions and several resource-duration modes, International Journal of Production Research 31 (11) (1993) 2547–2558.
[6] F.F. Boctor, Resource-constrained project scheduling by simulated annealing, International Journal of Production Research 34 (8) (1996) 2335–2351.
[7] K. Bouleimen and H. Lecocq. Multi-objective simulated annealing for the resource-constrained multi-project scheduling problem. in: Proceedings of the 7th International Workshop on Project Management and Scheduling, Osnabrück, Germany, 2000.
[8] P. Brucker, A. Drexl, R. Möhring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, models, and methods, European Journal of Operational Research 112 (1) (1999) 262–273.
[9] P. Brucker, S. Knust, A. Schoo, O. Thiele, A branch-and-bound algorithm for the resources-constrained project scheduling problem, European Journal of Operational Research 107 (1998) 272–288.
[10] N. Christofides, R. Alvarez-Valdes, J.M. Tamarit, Project scheduling with resource constraints: A branch-and-bound approach, European Journal of Operational Research 29 (2) (1987) 262–273.
[11] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, Management Science 38 (12) (1992) 1803–1818.
[12] E. Demeulemeester, W. Herroelen, New benchmark results for the resource-constrained project scheduling problem, Management Science 43 (11) (1995) 1485–1492.
[13] A. Drexl, J. Grünewald, Nonpreemptive multi-mode resource-constrained project scheduling, IIE transactions 25 (1993) 74–87.
[14] S. Hartmann, A competitive genetic algorithm for the resource-constrained project scheduling, Naval Research Logistics 456 (1998) 733–750.
[15] S. Hartmann, A. Drexl, Project scheduling with multiple-modes: A comparison of exact algorithms, Networks 32 (1998) 283–297.
[16] S. Hartmann, R. Kolisch, Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, European Journal of Operational Research 127 (2000) 394–407.
[17] W. Herroelen, E. Demeulemeester, B. De Reyck, A classification scheme for project scheduling, in: J. Weglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, 1999, pp. 1–26.
[18] W. Herroelen, E. Demeulemeester, B. De Reyck, Resource-constrained project scheduling—A survey of recent developments, Computers and Operations Research 25 (4) (1998) 279–302.
[19] U. Kohlmorgen, H. Schmeck, K. Haase, Experiences with fine-grained parallel genetic algorithms, Annals of Operations Research 90 (1999) 203–219.
[20] R. Kolisch, Efficient priority rule for the resource-constrained project scheduling problem, Journal of Operational Management 14 (1996) 179–192.
[21] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research 90 (1996) 320–333.
[22] R. Kolisch, A. Drexl, Adaptive search for solving hard scheduling problems, Naval Research Logistics 43 (1996) 23–40.
[23] R. Kolisch, A. Drexl, Local search for nonpreemptive multi-mode resource-constrained project scheduling, IIE Transactions 29 (1997) 987–999.
[24] R. Kolisch, S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computation analysis, in: J. Weglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer Academic Publishers, Boston, 1999, pp. 147–178.
[25] R. Kolisch, R. Padman, An integrated survey of project deterministic scheduling, OMEGA International Journal of Management Science 29 (3) (2001) 249–272.

[26] R. Kolisch, A. Sprecher, A. Drexl, Characterisation and generation of a general class of resource-constrained project scheduling problem, Management Science 41 (10) (1995) 1693–1703.
[27] J.K. Lee, Y.D. Kim, Search heuristics for the resource-constrained project scheduling problem, Journal of Operational Research Society 47 (1996) 678–689.
[28] A. Mingozzi, V. Maniezzo, S. Riccardelli, L. Bianco, An exact algorithm for project scheduling with resources constraints based on a new mathematical formulation, Management Science 44 (1998) 714–729.
[29] J.H. Patterson, A comparison of exact approaches for solving the multiple constrained resource project scheduling problem, Management Science 30 (7) (1984) 854–867.
[30] J.H. Patterson, R. Slowinski, F.B. Talbot, J. Weglarz, An algorithm for a general class of precedence and resource constrained scheduling problem, in: R. Slowinski, J. Weglarz (Eds.), Advances in Project Scheduling, Elsevier, Amsterdam, 1989, pp. 3–28.
[31] E. Pinson, C. Prins, F. Rullier, Using tabu search for solving the resource-constrained project scheduling problem, in: Proceedings of the 4th International Workshop on Project Management and Scheduling, Leuven, Belgium, 1994, pp. 102–106.
[32] A. Sprecher, Resource-constrained project scheduling: Exact methods for the multi-mode case, in: Lecture Notes in Economics and Mathematical Systems, vol. 409, Springer, Berlin, 1994.
[33] A. Sprecher, S. Hartmann, A. Drexl, An exact algorithm for project scheduling with multiple modes, OR Spektrum 19 (1997) 195–203.
[34] A. Sprecher, R. Kolisch, A. Drexl, Semi-active, active and non-delay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research 80 (1995) 94–102.