3-2006

A hybrid scatter search/electromagnetism meta-heuristic for project scheduling

Dieter DEBELS

Bert DE REYCK
Singapore Management University, bdreyck@smu.edu.sg

Roel LEUS

Mario VANHOUCKE

Citation

DEBELS, Dieter; DE REYCK, Bert; LEUS, Roel; and VANHOUCKE, Mario. A hybrid scatter search/ electromagnetism meta-heuristic for project scheduling. (2006). *European Journal of Operational Research*. 169, (2), 638-653.
Available at: https://ink.library.smu.edu.sg/lkcsb_research/6750

Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 169 (2006) 638–653
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

A hybrid scatter search/electromagnetism meta-heuristic for project scheduling

Dieter Debels [a,*], Bert De Reyck [b], Roel Leus [c], Mario Vanhoucke [a,d]

[a] *Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium*
[b] *London Business School, London, UK*
[c] *Department of Applied Economics, Katholieke Universiteit Leuven, Leuven, Belgium*
[d] *Operations and Technology Management Centre, Vlerick Leuven Gent Management School, Gent, Belgium*

Received 15 September 2003; accepted 18 May 2004

Abstract

In the last few decades, several effective algorithms for solving the resource-constrained project scheduling problem have been proposed. However, the challenging nature of this problem, summarised in its strongly *NP*-hard status, restricts the effectiveness of exact optimisation to relatively small instances. In this paper, we present a new meta-heuristic for this problem, able to provide near-optimal heuristic solutions for relatively large instances. The procedure combines elements from scatter search, a generic population-based evolutionary search method, and from a recently introduced heuristic method for the optimisation of unconstrained continuous functions based on an analogy with electromagnetism theory. We present computational experiments on standard benchmark datasets, compare the results with current state-of-the-art heuristics, and show that the procedure is capable of producing consistently good results for challenging instances of the resource-constrained project scheduling problem. We also demonstrate that the algorithm outperforms state-of-the-art existing heuristics.

Keywords: Project scheduling; Heuristics; Scatter search; Electromagnetism

1. Introduction

We study the resource-constrained project scheduling problem (RCPSP), denoted as $m,1|cpm|C_{\max}$ using the classification scheme of Herroelen et al. (1998a). The RCPSP can be stated as follows. A set of activities N, numbered from 0 to n ($|N| = n + 1$), is to be scheduled without

* Corresponding author.
E-mail addresses: dieter.debels@ugent.be (D. Debels), bdereyck@london.edu (B. De Reyck), roel.leus@econ.kuleuven.ac.be (R. Leus), mario.vanhoucke@ugent.be (M. Vanhoucke).

0377-2217/$ - see front matter
doi:10.1016/j.ejor.2004.08.020

pre-emption on a set R of renewable resource types. Activity i has a deterministic duration $d_i \in \mathbb{N}$ and requires $r_{ik} \in \mathbb{N}$ units of resource type k, $k \in R$, which has a constant availability a_k throughout the project horizon. We assume that $r_{ik} \leqslant a_k$, $i \in N$, $k \in R$. The dummy start and end activities 0 and n have zero duration while the other activities have a non-zero duration; the dummies also have zero resource usage. A is the set of pairs of activities between which a finish–start precedence relationship with time lag 0 exists. We assume graph $G(N, A)$ to be acyclic. A schedule S is defined by an $(n+1)$-vector of start times $\mathbf{s} = (s_0, \ldots, s_n)$, which implies an $(n+1)$-vector of finish times $\mathbf{e}(e_i = s_i + d_i, \forall i \in N)$. A schedule is said to be feasible if the precedence and resource constraints are satisfied. The objective of the RCPSP is to find a feasible schedule such that the schedule makespan e_n is minimised.

The research on the RCPSP has widely expanded over the last few decades, and reviews can be found in Brucker et al. (1999), Herroelen et al. (1998b), Icmeli et al. (1993), Kolisch and Padman (2001) and Özdamar and Ulusoy (1995). Numerous exact solution approaches have been advanced, with Brucker et al. (1998), Demeulemeester and Herroelen (1992, 1997), Mingozzi et al. (1998) and Sprecher (2000) perhaps the most noteworthy. However, the RCPSP, being a generalisation of the job shop scheduling problem, is strongly *NP*-hard (Blazewicz et al., 1983), and the computation times for exact algorithms can be excessive even for moderately sized instances. This has motivated numerous researchers to develop heuristic methods for dealing with RCPSP instances of practical sizes. Kolisch and Hartmann (1999) and Hartmann and Kolisch (2000) present a classification and performance evaluation of different such heuristics. Alcaraz and Maroto (2001) and Hartmann (1998, 2002) tackle the RCPSP by means of genetic algorithms, whereas Bouleimen and Lecocq (2003) use simulated annealing. Fleszar and Hindi (2004) implement a heuristic they refer to as "variable neighbourhood search". Nonobe and Ibaraki (2002) present a tabu search procedure that is applied to both the standard RCPSP problem as well as to various extensions. Palpant et al. (2004) discuss a large neighbourhood search approach in which blocks of activities scheduled within specific time slices are rescheduled by a commercial solver. Valls et al. (2003) present a population-based approach that adopts some tabu search principles. Valls et al. (2004) propose to use a combination of scatter search, path relinking, and improvement and solution combination procedures; their algorithm seems to be the best to date.

In this paper, we describe a new heuristic for the RCPSP, inspired by recent advances in the development of meta-heuristics. The procedure combines elements from scatter search (SS), a population-based evolutionary search method, and a recently introduced heuristic method for the optimisation of unconstrained continuous functions that simulates the electromagnetism theory of physics, hereafter referred to as the electromagnetism (EM) meta-heuristic. We extend the EM heuristic for combinatorial optimisation problems and integrate it into an SS framework. In Section 2, we explain how we represent and evaluate RCPSP solutions. Our search strategy is cast into an SS framework, as outlined in Section 3. Section 4 describes the main elements of the EM heuristic applied to unconstrained continuous optimisation. In Section 5, we show how the EM methodology can be modified to be used in a combinatorial optimisation setting and how it can be combined with an SS algorithm for the RCPSP. Section 6 discusses an intensification strategy that is used to enhance the effectiveness and efficiency of the algorithm. Section 7 contains the computational results on benchmark datasets, as well as a comparison with other current state-of-the-art heuristics. We conclude with Section 8.

2. Representation, schedule generation and solution evaluation

The backbone of most improvement heuristics for solving the RCPSP, where an initial (set of) solution(s) is gradually improved, is a *schedule representation* scheme, a *schedule generation* scheme and a *solution evaluation* procedure. Typically, an RCPSP improvement heuristic does not operate directly on a schedule, but on some representation of a schedule that is convenient and effective for

the functioning of the algorithm. After an operation on a solution (i.e. on a schedule represented in a particular way) has been performed, the newly obtained solution is transformed into a schedule using a schedule generation scheme (SGS). We will use a similar approach in this paper.

Kolisch and Hartmann (1999) distinguish between various representations for schedules in the development of heuristics for the RCPSP. The two most important ones are the *random-key (RK)* representation and the *activity-list (AL)* representation. In RK form, a solution corresponds to a point in Euclidian $(n+1)$-space, such that the ith vector element functions as a priority value for the ith activity. Using a serial schedule generation scheme, these priority values can then be used to construct an active schedule by scheduling each activity one-at-a-time and as early as possible within the precedence and resource constraints. Alternatively, a parallel SGS could be used, although Kolisch (1996) has shown that, contrary to the serial SGS, the parallel SGS is sometimes unable to reach an optimal solution. In the AL representation, a schedule is represented by a linear extension of the partial order induced by the precedence constraints, such that an SGS gives priority to the activity that comes first in the list containing a complete order on N. This is similar to list scheduling in machine scheduling.

Hartmann and Kolisch (2000) report that in general, procedures that make use of the AL representation perform better than those based on the RK form. This claim is based solely on computational tests, and no underlying reasons are cited. We believe that the main reason for the inferior performance of the RK representation lies in the fact that one single schedule can have many different representations. This results in a larger solution space, and the problem that the structure of a solution or schedule representation does not necessarily contain information about the quality of the associated schedule, which sometimes prevent (meta-) heuristics operating on schedule representations from making improvements. The AL representation also suffers from this, in that a single schedule can be represented by different activity lists. This problem, however, occurs more frequently using the RK representation, for reasons we will explain below.

Despite this disadvantage, the RK representation has the advantage that each solution corresponds to a point in Euclidian $(n+1)$-space, so that geometric operations can be performed on its components. Since this is one of the cornerstones of both the SS and EM methods, we adopt the RK representation, allowing us to perform mathematical operations on solutions. We have modified the standard RK representation in order to eliminate the problem stated above, thereby removing its comparative disadvantage relative to the AL form.

There are four underlying reasons why a schedule can be represented by different RK forms, caused by (1) scaling, (2) precedence constraints, (3) timing anomalies and (4) activities with identical starting times. We will discuss these problems one by one and show how these problems can be eliminated using a unique, standardised form of the RK representation. Note that problems (3) and (4) also occur for activity lists. By eliminating all four problem areas, our unique RK representation will perform better than both the standard RK as well as the standard AL forms.

We introduce the example project depicted in Fig. 1, with a single renewable resource type with availability $a_1 = 2$. A feasible schedule for this scheduling problem, with a makespan equal to 18, is given in Fig. 2. Assuming that lower RK values correspond to higher priorities, the schedule in Fig. 2 can be obtained with the following RK vector: $\mathbf{x}_1 = [0.9; 1.1; 2.6; 0.7; 2.1; 0.8; 1.0; 1.9; 3.2]$ (we omit the RK values for the dummy start and end activity).

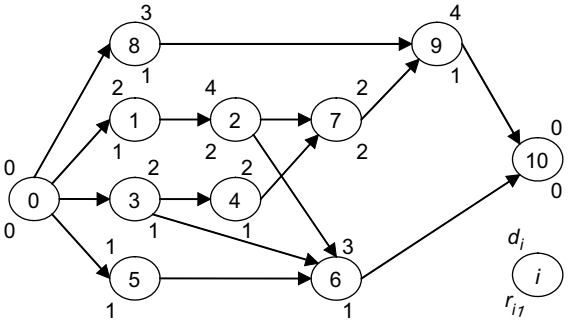

Fig. 1. Example project.

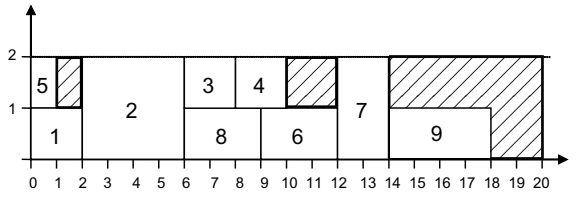

Fig. 2. A schedule for the example project.

(1) Scaling in Euclidean space

Scaling the priority values of any RK representation up or down results in a different RK representation, which, however, will always result in the same schedule. In fact, there exist an infinite number of RK representations with different priority values, but with the same priority structure. For our example, $\mathbf{x}_2 = [10; 14; 30; 6; 25; 7; 11; 20; 35]$ results in the same schedule. We eliminate this problem by replacing the priority values by their rank values. For the example, we can transform $\mathbf{x}_1$ or $\mathbf{x}_2$ into $\mathbf{x}_3 = [3; 5; 8; 1; 7; 2; 4; 6; 9]$, which also yields the schedule in Fig. 2.

(2) Precedence constraints

In an RK representation, priority values are not constrained by the precedence constraints, in the sense that the RK of an activity can be higher than the RK of one of its predecessors. Essentially, this is not a problem since an SGS will take the precedence relations into account, but it can lead to different RK representations for a single schedule. In our example, we can see that activity 7 has a higher priority in $\mathbf{x}_3$ (a lower RK) than activities 2 and 3, two predecessors of activity 7. A serial SGS would schedule the activities in the following order: 1, 2, 8, 5, 3, 4, 6, 7, 9, i.e. taking into account the precedence relations. However, another RK representation such as $\mathbf{x}_4 = [5; 6; 9; 2; 8; 3; 4; 7; 1]$ would result in the same schedule. To eliminate this problem, we set the RK values of each activity equal to its rank order in the activity list obtained using a serial SGS. This results in an RK representation with priority values "in line" with the precedence constraints. For our example, we obtain $\mathbf{x}_5 = [1; 2; 5; 6; 4; 7; 8; 3; 9]$.

(3) Timing anomalies

The previous two problems arise only with the RK representation. There are two more problems, associated with both the RK and AL representation. The first is caused by the following phenomenon. If an activity a_1 starts earlier than another activity a_2 in a schedule, then an AL representation of this schedule exists with a_1 having a higher priority than a_2. If, however, none of the activities starting after a_1 and before a_2 in the activity list, nor a_2 itself, could be scheduled earlier if activity a_1 were removed from the schedule (because of precedence and/or resource constraints), then there also exists an AL representation for the same schedule in which a_1 has a lower priority than a_2.

In the example schedule of Fig. 2, activity 5 starts earlier than activity 8. Therefore, there is an AL in which activity 5 has higher priority than activity 8. Nevertheless, in $\mathbf{x}_5 = [1; 2; 5; 6; 4; 7; 8; 3; 9]$, which also leads to the schedule in Fig. 2, the RK of activity 5, namely 4, is higher than the RK of activity 8, namely 3, and thus activity 5 receives lower priority. This is due to the fact that even in the absence of activity 5, activity 8 cannot be scheduled earlier due to activities 1 and 2 requiring a significant amount of the resource in periods 1–6. Activity 5, consuming fewer resources and taking less time, can be inserted into the schedule at time 0 both before and after activity 8 is included. In other words, there are at least two priority vectors leading to the same schedule.

To deal with this problem, we propose to use a topological order (TO) representation of schedules (Valls et al., 2003, 2004): for a schedule S, a TO representation of S is any vector $\mathbf{x}$ containing the numbers from 0 to $n+1$ and for which $s_i(S) < s_j(S)$ implies $x_i < x_j$. Adhering to the TO representation eliminates the problem discussed above. For the example schedule in Fig. 2, activity 5 receives the second highest priority in the TO representation. Consequently, $\mathbf{x}_5$ is replaced by $\mathbf{x}_6 = [1; 3; 5; 6; 2; 7; 8; 4; 9]$.

(4) Activities with the same starting times

Even with the TO representation, there can still be multiple representations of a single schedule. If two activities i and j start at the same time, the priorities of i and j can be interchanged without affecting the associated schedule. In $\mathbf{x}_6$, this is the case for activities 1 and 5, and 3 and 8, respectively. To prevent this, we attribute the lowest ranking

to *all* activities starting at the same time. By doing so for the example schedule, we end up with $\mathbf{x}_7 = [1;3;4;6;1;7;8;4;9]$, a unique standardised random key (SRK) representation for the schedule.

Using the SRK schedule representation, each solution in SRK form is uniquely associated with a schedule. When in our algorithm new solutions are created, which do not necessarily conform to the SRK form, they are transformed into SRK form while at the same time evaluating the associated objective function value as follows. When a new priority vector $\mathbf{x} \in \mathbb{R}^{n+1}$ is obtained, we compute a schedule $S = \sigma(\mathbf{x})$, using a SGS σ, with associated objective function value equal to the makespan $e_n(\sigma(\mathbf{x}))$. We then replace $\mathbf{x}$ by SRK priority vector $\pi(\sigma(\mathbf{x}))$, where π transforms the schedule to SRK-standardised priorities, based on the activity starting times in $\sigma(\mathbf{x})$. In this way, when we work with a population of solutions, we guarantee that each solution corresponds to a unique schedule.

3. Scatter search

Scatter search (SS) is an evolutionary or population-based method in which solutions are combined to yield better solutions using convex or non-convex linear combinations. Strategies for diversification and intensification are typically added to enhance the search. SS contrasts with other evolutionary procedures such as genetic algorithms (GA) by providing unifying principles for joining solutions based on generalised path constructions in Euclidean space and by utilizing strategic designs where other approaches resort to randomisation. Some sources (for instance Taillard et al., 2001) see SS as a very generic methodology, constituting a generalisation of the GA procedure. For a general introduction to SS, we refer the reader to Glover et al. (2000, forthcoming) and Martí et al. (2004). The algorithm we present in this paper for solving the RCPSP contains a SS skeleton as outlined in pseudo-code below:

```
Algorithm SS

1. construct a pool P of randomly generated solutions
2. construct RefSet = B1 ∪ B2, B1 ∪ B2 ⊂ P, B1 ∩ B2 = ∅
while (nr_schedules < schedule_limit) do
   3. generate subsets from RefSet
   4. create a new pool P of trial solutions by applying a solution combination method to each subset
   5. update RefSet
endwhile
```

In step 1, a large pool P of solutions is generated: solution vectors are obtained by randomly generating each of their components; these are transformed to SRK format when the schedule is evaluated, in the way described in the previous section. In step 2, a reference set *RefSet* is constructed from P containing high quality solutions (subset B_1) and diverse solutions (subset B_2), with $B_1 \cap B_2 = \emptyset$; this "two-tier" design is maintained throughout the search. B_1 contains the b_1 solutions in P with best makespan, while a threshold t_1 on the minimal distance between the elements in B_1 is imposed in pursuit of diversity. B_2 contains the best (minimum makespan) schedules from $P \backslash B_1$ that are sufficiently distant from the elements of B_1. The latter diversity requirement is achieved by means of a threshold t_2 on the smallest distance to any element in B_1 with $t_2 > t_1$. As a distance measure, we use the sum of the absolute values of the component-wise differences divided by the number of activities. As a stopping criterion, we impose a limit on the number of generated schedules, which is in line with the existing literature on RCPSP heuristics.

In steps 3 and 4, a new pool of solutions is created using a solution combination method to pairs of solutions in *RefSet*. This is performed in two different ways:

1. *Pairs in* B_1: All pairs in B_1 containing at least one new solution compared to the previous generation are considered. From each such pair, two children are produced by means of a standard two-point crossover operator, which are added to the new pool.
2. *Elements from* $B_1 \times B_2$: From each combination of one element from B_1 and one from B_2 a single

offspring is constructed by means of EM (see Sections 4 and 5).

In step 5, *RefSet* is updated on the basis of the newly generated pool of solutions. We opt for a *static* update, in which the reference set is updated only after the new pool is completely generated. B_1 is updated by considering also the new solutions in the pool; when there is a tie, preference is given to new solutions. B_2 is recomputed as the set of remaining minimum makespan solutions that are sufficiently distant from the elements of B_1.

4. The electromagnetism meta-heuristic

Birbil and Fang (2003) propose a so-called electromagnetism (EM) optimisation heuristic for unconstrained global optimisation problems, i.e. the minimisation of non-linear functions. Convergence details for the heuristic are provided in Birbil et al. (forthcoming). In a multi-dimensional solution space where each point represents a solution, a *charge* is associated with each point. This charge is related to the objective function value associated with the solution. As in evolutionary search algorithms, a population, or set of solutions, is created, in which each solution point will exert attraction or repulsion on other points, the magnitude of which is proportional to the product of the charges and inversely proportional to the distance between the points (Coulomb's Law). The principle behind the algorithm is that inferior solution points will prevent a move in their direction by repelling other points in the population, and that attractive points will facilitate moves in their direction. This can be seen as a form of local search in Euclidian space in a population-based framework. The main difference with existing methods is that the moves are governed by forces that obey the rules of electromagnetism. Birbil and Fang (2003) provide a generic pseudo-code for the EM algorithm:

```
Algorithm EM

while (stopping criterion not met) do
  1. local search
  2. compute forces
  3. apply forces
endwhile
```

Step 1 explores the immediate (Euclidian) neighbourhood of individual points in the population. The total force exerted on each point by all other points is calculated in step 2. This force depends on the charge of the point under consideration as well as of the points exerting the force. The charge of each point $\mathbf{x}_i$ is determined by its objective function value $f(\mathbf{x}_i)$ in relation to the objection function value of the current best point $\mathbf{x}^{\text{best}}$ in the population, with better objective function values resulting in higher charges. For a minimisation problem, the charge q_i of point $\mathbf{x}_i$ is determined according to Eq. (4.1):

$$q_i = \exp\left(-d\frac{f(\mathbf{x}_i) - f(\mathbf{x}^{\text{best}})}{\sum_{k=1}^{m}(f(\mathbf{x}_k) - f(\mathbf{x}^{\text{best}}))}\right). \tag{4.1}$$

The parameter m represents the population size, d is the dimension of the solution space. Subsequently, a set of force vectors $\mathbf{F}_i$, $i = 1,\ldots,m$, is determined, that are exerted on points $\mathbf{x}_i$:

$$\mathbf{F}_i = \sum_{\substack{j=1\\ j\neq i}}^{j=m} \begin{cases} (\mathbf{x}_j - \mathbf{x}_i)\left(\frac{q_i q_j}{\|\mathbf{x}_j - \mathbf{x}_i\|^2}\right) \\ \quad \text{if } f(\mathbf{x}_j) < f(\mathbf{x}_i) \\ (\mathbf{x}_i - \mathbf{x}_j)\left(\frac{q_i q_j}{\|\mathbf{x}_j - \mathbf{x}_i\|^2}\right) \\ \quad \text{if } f(\mathbf{x}_j) \geqslant f(\mathbf{x}_i) \end{cases}. \tag{4.2}$$

A point with a relatively good objective function value attracts the other ones, points with inferior objective value repel the other population members. The forces exerted on i by all other points are combined by means of vector summation, as shown in the two-dimensional example in Fig. 3. $\mathbf{F}_{13}$ is the force exerted by $\mathbf{x}_1$ on $\mathbf{x}_3$ (repulsion: the objective function value of $\mathbf{x}_1$ is worse than that of $\mathbf{x}_3$) and $\mathbf{F}_{23}$ is the force exerted by $\mathbf{x}_2$ on $\mathbf{x}_3$ (attraction: the objective function value of $\mathbf{x}_2$ is better than that of $\mathbf{x}_3$). The total force exerted on $\mathbf{x}_3$ equals $\mathbf{F}_3 = \mathbf{F}_{13} + \mathbf{F}_{23}$.

Movement according to the resulting forces results in a new population of solutions. Contrary to the simplified example in Fig. 3, the imposed force

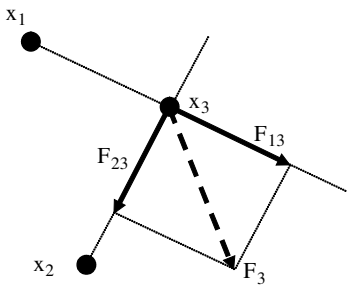

Fig. 3. Example of exertion of forces.

is normalised by division by its norm, and therefore only identifies the direction of the move, not the magnitude. The magnitude of each move is determined for each dimension separately, and is equal to a value randomly selected from domain $[0; maxmove]$, where *maxmove* indicates the maximum allowable movement in the particular dimension.

5. Modifying the EM algorithm for the RCPSP

In our SS procedure, part of the combination method to create a new pool of solutions from the solutions in *RefSet* is implemented using the EM framework. In the following section, we will discuss how we have extended the EM methodology for combinatorial optimisation and for the RCPSP in particular, and how it can be integrated into a general SS framework.

In the basic EM algorithm, all points in a population exert a force on all other points. We generalise this concept by allowing a pre-determined number $z \in [1; m-1]$ of points to act on any given point, where m is the population size. Experiments have shown that the choice $z = 1$ yields good results and can be easily implemented, so that we have restricted our procedure to this value. Recall that in step 3 of our algorithm, we create a new pool of solutions by combining pairs of activities in $RefSet = B_1 \cup B_2$. For all pairs in $B_1 \times B_2$ a force is exerted by the point in B_1 on the point in B_2 attracting it in its direction. For all pairs from $B_1 \times B_1$ however, we use a crossover operator rather than an EM movement because the makespan values of the corresponding schedules will tend to be very similar, which reduces the effectiveness of the EM algorithm as it looks for improvements based on differences in objective function values to guide the search.

For determining the force exerted on point i (from B_2) by point j (from B_1), we do not use fixed charges q_i and q_j as in the standard EM algorithm, but instead a charge q_{ij} that depends on the relative difference in objective function value between i and j. So, contrary to the basic EM algorithm, point charges are not computed independently but based on the point they exert force on

$$q_{ij} = \frac{f(\mathbf{x}_i) - f(\mathbf{x}_j)}{f(\mathbf{x}^{\text{worst}}) - f(\mathbf{x}^{\text{best}})} \tag{5.1}$$

with $\mathbf{x}^{\text{worst}}$ and $\mathbf{x}^{\text{best}}$ the worst and best solutions in *RefSet*. As a result, $q_{ij} \in [-1;1]$ and "better" points j have higher scores on q_{ij}. More specifically, if $f(\mathbf{x}_i) > f(\mathbf{x}_j)$, i.e. when point i has a higher makespan than point j, q_{ij} is positive and j attracts i. The opposite, i.e. repulsion, occurs when $f(\mathbf{x}_i) < f(\mathbf{x}_j)$, and no action is taken when $f(\mathbf{x}_i) = f(\mathbf{x}_j)$.

In our implementation, the force exerted by point j on point i equals

$$\mathbf{F}_{ij} = (\mathbf{x}_j - \mathbf{x}_i) \cdot q_{ij}. \tag{5.2}$$

We then move from solution $\mathbf{x}_i$ to $\mathbf{x}_i + \mathbf{F}_{ij}$ in the direction of $\mathbf{x}_j$, but $\mathbf{x}_j$ itself is rarely reached because the multiplier q_{ij} in the right-hand side of (5.2) is almost always smaller than 1. Based on the computed force and the resulting movement, new solutions are created in Euclidian space. In a sense, this method is similar to the meta-heuristic *path relinking* (Glover et al., 2000), which is based on gradual moves from one solution in the direction of another. EM offers a generic framework to determine these movements.

In the basic EM algorithm, forces are exerted in each dimension. For the RCPSP, this corresponds to a change in the priority of each activity. We extend this idea by allowing forces to act only in a particular subset of the dimensions. We randomly select $p_{\min} \in [1; n-1]$ and $p_{\max} \in [2; n]$ with $p_{\min} \leqslant p_{\max}$ and a minimum distance between $p_{\min}$ and $p_{\max}$. We update only the SRK values between $p_{\min}$ and $p_{\max}$ (inclusive) according to the forces exerted in these dimensions. Note that due to the SRK

representation, the thus updated activities all start within a particular time interval. The other SRK components are updated as follows. We subtract a large constant (n or higher) from all SRK values lower than $p_{\min}$ and add the same constant to all SRK values higher than $p_{\max}$. This preserves the priority structure of the activities unaffected by the forces, and the relative priorities of the three corresponding subsets of activities. For the activities with SRK value $<p_{\min}$ or $>p_{\max}$ the relative priorities are taken from the original solution, and remain the highest priorities (when SRK value $< p_{\min}$) or lowest priorities (when SRK value $> p_{\max}$), compared to the activities with SRK value between $p_{\min}$ and $p_{\max}$ for which the new priority values are determined using EM. In fact, this means that we have hybridised the basic EM move with a two-point crossover, which results in increased flexibility of the algorithm and improved performance. Note that the resulting RK vector is not necessarily in SRK form, but can be transformed into SRK format.

We again consider the example project presented in Section 2, and the schedule associated with SRK vector $\mathbf{x}_8 = [3;6;1;4;1;7;8;5;9]$, which is depicted in Fig. 4. The makespan of this schedule equals $e_n(\sigma(\mathbf{x}_8)) = 19$. As explained in Section 3, $\mathbf{x}_8$ will be combined with the elements in B_1 using EM. Consider the schedule given in Fig. 5 with a makespan equal to 15 and RK representation $\mathbf{x}_9 = [1;3;1;4;6;8;7;4;8]$.

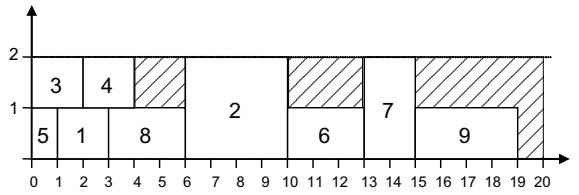

Fig. 4. A schedule for the example project (associated with vector x_8).

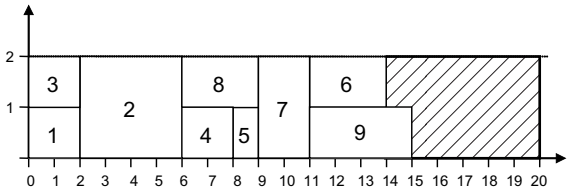

Fig. 5. The schedule $\sigma(x_9)$.

We will illustrate the functioning of the EM solution combination method by examining the effects of a force exerted by $\mathbf{x}_9$ on $\mathbf{x}_8$, with $p_{\min} = 3$ and $p_{\max} = 7$. We assume that $f(\mathbf{x}^{\text{worst}}) = 22$ and $f(\mathbf{x}^{\text{best}}) = 15$, so that the charge $q_{89} = 4/7 \approx 0.57$. Force $\mathbf{F}_{89}$ on $\mathbf{x}_8$ can now be computed as $q_{89}(\mathbf{x}_9 - \mathbf{x}_8)$. In Table 1, the components of $\mathbf{x}_8$ in interval $[p_{\min};p_{\max}]$ are bolded and the vector added to $\mathbf{x}_8$ is referred to as $\mathbf{F}'_{89}$, with $\mathbf{x}_8 + \mathbf{F}'_{89} = \mathbf{x}'_8$. Finally, we can transform $\mathbf{x}'_8$ into its SRK form $\mathbf{x}''_8 = \pi(\sigma(\mathbf{x}'_8))$. The associated schedule $\sigma(\mathbf{x}'_8) = \sigma(\mathbf{x}''_8)$ is depicted in Fig. 6, with $e_n(\sigma(\mathbf{x}''_8)) = 17$.

Note that the implemented hybrid two-point/EM move will not simply copy the priority structure of $\mathbf{x}_9$ for the part of the vector between $p_{\min}$ and $p_{\max}$ but rather result in a priority structure that is somewhere between $\mathbf{x}_8$ and $\mathbf{x}_9$. A standard two-point crossover can be implemented as follows:

1. SRK $< p_{\min}$: a large constant value (e.g. n) is subtracted from the priority value.

Table 1
Illustration of the execution of a move

Activities	1	2	3	4	5	6	7	8	9
$\mathbf{x}_8$	**3**	**6**	1	**4**	1	**7**	8	*5*	9
$\mathbf{x}_9$	1	3	1	4	6	8	7	4	8
$\mathbf{F}_{89}$	−1.14	−1.71	0	0	2.86	0.57	−0.57	−0.57	−0.57
$\mathbf{F}'_{89}$	−1.14	−1.71	−10	0	−10	0.57	+10	−0.57	+10
$\mathbf{x}'_8$	1.86	4.29	−9	4	−9	7.57	18	4.43	19
$\mathbf{x}''_8$	3	5	1	4	1	6	8	6	9

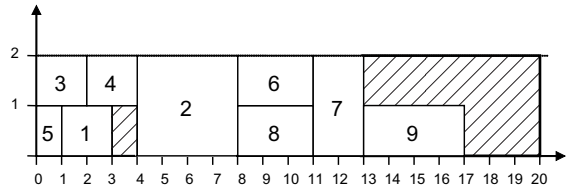

Fig. 6. The resulting schedule after execution of a move.

2. $p_{\min} \leqslant \mathrm{SRK} \leqslant p_{\max}$: copy the attracting vector.
3. $p_{\max} < \mathrm{SRK}$: a large constant value is added to the priority value.

For $\mathbf{x}_8$ and $\mathbf{x}_9$ above, such a crossover would lead to $\mathbf{x}_{10} = [1;3;-9;4;-9;8;18;4;19]$, with $e_n(\sigma(\mathbf{x}_{10})) = 18$ and $\pi(\sigma(\mathbf{x}_{10})) = [3;4;1;5;1;7;8;5;9]$. In other words, a two-point crossover yields a schedule with makespan 18, whereas the hybrid move results in a schedule with makespan 17. The second crossover offspring, which takes its lowest and highest priorities from $\mathbf{x}_9$ and its middle part from $\mathbf{x}_8$, reproduces $\sigma(\mathbf{x}_8)$ with makespan 19.

6. Intensification

The makespan $e_n(\sigma(\mathbf{x}))$ associated with a solution $\mathbf{x}$ is obtained using a serial SGS σ. In order to improve the intensification characteristics of the algorithm, we use an enhanced generation scheme σ^* that iteratively looks for improvements in the priority vector using global forward and backward shifts of individual activities. The scheme σ^* guarantees that $e_n(\sigma^*(\mathbf{x})) \leqslant e_n(\sigma(\mathbf{x}))$, and results in a standardised solution $\pi(\sigma^*(\mathbf{x}))$. Our method is based on principles described by Li and Willis (1992) and Özdamar and Ulusoy (1996).

First we apply σ on $\mathbf{x}$, yielding an active, i.e. left-justified, schedule. Next, we iteratively perform backward and forward passes. The priorities used in these passes are based on the SRK vector consisting of the ending times (backward) and starting times (forward) in the schedule, which results in a right-justified and left-justified schedule, respectively. The schedule makespan of each intermediate schedule is never higher than the makespan of the previous one. In this way, we exploit opportunities for global right and left shifts of individual activities in order to reduce the makespan.

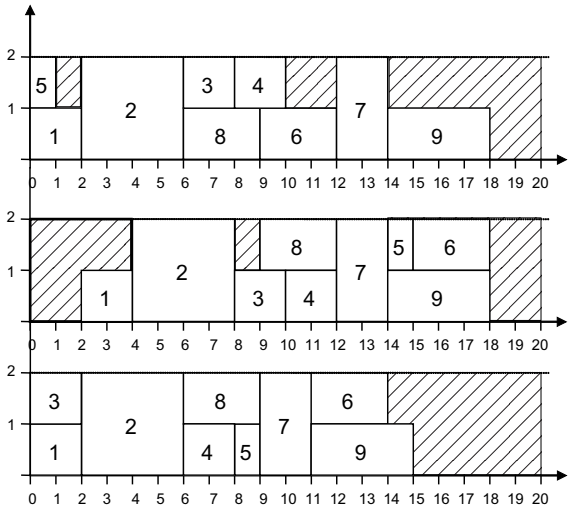

Fig. 7. Stepwise improvement of the makespan.

A computational example will illustrate our approach: consider the project of Fig. 1 and RK vector $\mathbf{x}_7 = [1;3;4;6;1;7;8;4;9]$. $\sigma(\mathbf{x}_7)$ was depicted in Fig. 2 and is repeated at the top of Fig. 7. The schedule has a makespan of 18 time units. We now try to reduce the makespan by shifting each activity, in decreasing order of activity end times, as much as possible to the right without affecting the project completion time. Activities 9 and 7 cannot be scheduled later. Activity 6 can be right shifted to start at time 15. Activity 4 can be shifted two time units and start at time 10. Since the global right shift of activity 6 has made some additional resources available, activity 8 can be shifted three time units to start at time instant 9. Activities 3, 2 and 1 can shift two time units. Finally, activity 5 is shifted to time 14. In this way, we obtain a schedule with a makespan of 16 units. Further improvements of the schedule are possible by shifting activities as much as possible to the left. This reduces the makespan by one further time unit, as illustrated in Fig. 7. This procedure is continued until no further improvements can be found.

7. Computational experiments

We have coded the procedure in Visual C++ 6.0 and performed computational tests on an Acer Travelmate 634LC with a Pentium IV 1.8 GHz processor using two different testsets. The first set is composed of instances generated by RanGen (Demeulemeester et al., 2003) and is used to study the impact of the different parameters on the performance of the algorithm, and to determine the optimal settings for these parameters. The second testset is the well-known PSPLIB dataset (Kolisch and Sprecher, 1997), used to report computational results of our procedure and to compare with other state-of-the-art results.

7.1. Impact of the parameters

To test the impact of the different parameters on the effectiveness and efficiency of the procedure, we have constructed a dataset containing 480 instances using RanGen (Demeulemeester et al., 2003). Each instance contains 75 activities and has been generated with the following settings. The order-strength is set at 0.25, 0.50 or 0.75, resource usage at 1, 2, 3 or 4 and the resource-constrainedness at 0.2, 0.4, 0.6 or 0.8. Using 10 instances for each problem class, we obtain a problem set with 480 network instances.

This approach is similar to the way Valls et al. (2003, 2004) derive their computational results. The authors optimise the values of the different parameters based on a subset of the J120 instances, and then test the effectiveness of the algorithm on the complete testset. Although the results would be improved by optimising the parameter values for the entire testset, the approach by Valls et al. (2003, 2004) is more suitable since the results do not rely on customising the parameters for that particular set. We opt for a similar approach, but take it one step further by not optimising the parameter values on the testset at all, not even on a subset, but on a completely different testset as described in this section.

The size of the initial solution pool P is set to 100. Table 2 illustrates the influence of the parameters b_1 and b_2 i.e. the number of solutions in subset B_1 and B_2 on the performance of the algorithm for 1000, 5000, 50,000 and 500,000 schedules, respectively. The column "Sum" contains the sum of the 480 project makespans, and the column "Avg. Dev." contains the average deviation from the critical-path lower bound. The table reveals that the optimal values for b_1 and b_2 (with

Table 2
Impact of parameters b_1 and b_2

	Sum	Avg. Dev. (%)	Sum	Avg. Dev. (%)	Sum	Avg. Dev. (%)
1000 Schedules	$b_1 = 4$		$b_1 = 5$		$b_1 = 6$	
$b_2 = 2$	91,659	278.9	91,632	278.8	91,635	278.8
$b_2 = 3$	91,637	278.8	**91,612**	**278.8**	91,726	279.3
$b_2 = 4$	91,634	278.9	91,691	279.1	91,681	279.1
5000 Schedules	$b_1 = 8$		$b_1 = 10$		$b_1 = 12$	
$b_2 = 4$	91,032	276.2	90,930	275.7	90,966	275.9
$b_2 = 5$	90,951	275.8	**90,920**	**275.7**	90,965	275.9
$b_2 = 6$	90,945	275.8	90,938	275.9	90,927	275.8
50,000 Schedules	$b_1 = 26$		$b_1 = 28$		$b_1 = 30$	
$b_2 = 14$	90,457	273.6	90,454	273.6	90,471	273.7
$b_2 = 16$	90,456	273.6	**90,433**	**273.5**	90,443	273.6
$b_2 = 18$	90,452	273.6	90,441	273.6	90,449	273.6
500,000 Schedules	$b_1 = 62$		$b_1 = 65$		$b_1 = 68$	
$b_2 = 30$	90,215	272.6	90,226	272.5	90,230	272.5
$b_2 = 33$	90,225	272.5	**90,209**	**272.5**	90,211	272.5
$b_2 = 36$	90,225	272.5	90,217	272.5	90,210	272.5

solutions highlighted in bold) depend on the number of schedules. Not all test results are shown, only the ones surrounding the parameter values that were found to be optimal. The tests also revealed that the optimal values for parameters t_1 and t_2 (the diversity thresholds for B_1 and B_2) were 1.1 and 2.0, respectively, and were not found to be sensitive to the schedule limit.

7.2. Comparative results with best known solutions

In order to compare with the best results from the literature, we use the well-known J30, J60, J90 and J120 instances of the PSPLIB testset (Kolisch and Sprecher, 1997). Table 3 shows the results that were obtained using the optimal parameter settings for the RanGen set as described in the previous section. The row labelled "Sum" contains the sum of the makespans of all problem instances. The row labelled "Avg. Dev. CPM" reports the average deviation from the critical-path lower bound. The row labelled "Avg. Dev. Best" displays the average percentage deviation from the currently best known solution in PSPLIB as reported on September 12, 2003. For the J30 set these solutions are all optimal. The fourth row, labelled "Best", shows the number of instances for which our heuristic algorithm reports a makespan equal to the currently best solution. The fifth row, labelled "Improved", reports the number of problem instances for which we have been able to improve the best known solution (based on PSPLIB

Table 3
Computational results

Problem set	Schedules	J30	J60	J90	J120
Sum	1000	28,410	38,765	46,321	76,736
	5000	28,355	38,554	46,030	75,537
	50,000	28,319	38,420	45,822	74,671
	500,000	28,319	38,357	45,700	74,055
Avg. Dev. CPM	1000	13.77%	11.73%	11.30%	35.22%
	5000	13.54%	11.10%	10.59%	33.10%
	50,000	13.38%	10.71%	10.09%	31.57%
	500,000	13.38%	10.53%	9.80%	30.48%
Avg. Dev. Best	1000	0.27%	0.90%	1.11%	3.24%
	5000	0.11%	0.46%	0.61%	1.90%
	50,000	0.01%	0.19%	0.27%	0.95%
	500,000	0.01%	0.07%	0.07%	0.28%
Best	1000	421 (480)	360 (480)	365 (480)	199 (600)
	5000	451 (480)	386 (480)	373 (480)	225 (600)
	50,000	477 (480)	415 (480)	395 (480)	282 (600)
	500,000	477 (480)	447 (480)	437 (480)	434 (600)
Improved	1000	–	0	0	0
	5000	–	0	0	0
	50,000	–	0	1	3
	500,000	–	2	13	35
Avg. CPU (seconds)	1000	0.01	0.03	0.07	0.12
	5000	0.06	0.18	0.37	0.65
	50,000	0.69	1.88	4.03	6.66
	500,000	7.16	19.61	38.87	69.57
Max. CPU (seconds)	1000	0.05	0.07	0.13	0.22
	5000	0.12	0.27	0.70	0.93
	50,000	1.27	2.64	8.02	9.22
	500,000	9.89	33.14	54.94	98.45

Table 4
Optimal set-dependent parameter settings

	Problem set	RanGen	J30	J60	J90	J120
b_1	1000 Schedules	5	6	4	5	4
	5000 Schedules	10	10	10	8	8
	50,000 Schedules	28	28	26	20	26
b_2	1000 Schedules	3	4	4	2	2
	5000 Schedules	5	3	7	5	5
	50,000 Schedules	16	16	16	14	12

results on February 9, 2004, see http://www.bwl.uni-kiel.de/bwlinstitute/Prod/psplib/datasm.html). The last rows, labelled "Avg. CPU" and "Max. CPU", indicate the average and maximal computation time to solve a problem instance. Each cell of the table displays the results for a run with maximum 1000, 5000, 50,000 and 500,000 schedules.

The results indicate that the algorithm is capable of providing near-optimal solutions for set J30 within very small computation times, and competitive solutions for the other problem sets, all with limited computational effort. Also, the results show only a moderate increase in required computational effort when the problem size increases, which is an encouraging result since this should allow for solving very large scale instances. We have also been able to improve the best known solutions for several of the problem instances in the PSPLIB set. During our experiments with the algorithm, we have been able to find better solutions for 6 instances of the J60 set, 44 instances of the J90 set and 99 instances of the J120 set. Since these best known solutions have been obtained using a large set of solution procedures, including exact ones for some sets, these improvements are another indication of the potential usefulness of the proposed heuristic.

When the parameters settings are optimised for these problem sets, a slight improvement in the performance of the heuristic can be observed. The biggest improvement was found for J120, where the average deviation from the lower bound could be reduced from 35.22% to 34.90% for 1000 schedules. However, overall, the optimal parameter settings for each set are quite close to those determined using the RanGen set, which shows that these settings are robust with respect to changes in the problem characteristics (see Table 4). For the comparative computational experiments in the following sections, the original parameter settings (i.e. not optimised for a particular set) are used.

7.3. Comparative results with 5000 schedule limit

In the following tables we provide a comparison with the best heuristic procedures reported in the literature. In order to have a fair base of comparison, we only compare the results with a limit of 5000 schedules, and omit procedures that do not report such results (these will be discussed later). To measure the effectiveness of the algorithms, we report the average deviation of the heuristic solutions from the critical path, except for J30, where we report the average deviation from the optimal solution. We also provide a rank order of effectiveness for each problem set in column "*R*". Empty cells denote that, to the best of our knowledge, no results are stated. Table 5 reveals that our new algorithm performs consistently well over all problem sets, and outperforms the currently best performing procedure in each class.

7.4. Comparative results with extended time limit

In this section we provide a comparison with other state-of-the-art heuristics for which computational results with a limited number of schedules are not available. These include Valls et al. (2003, 2004), Fleszar and Hindi (2004) and Palpant et al. (2004). We also compare with results obtained by the algorithm of Nonobe and Ibaraki (2002) without a limit on the number of schedules (as reported

Table 5
Comparative computational results with limit on number of schedules

Author	Problem set							
	J30		J60		J90		J120	
	Dev. (%)	*R*	Dev. (%)	*R*	Dev. (%)	*R*	Dev. (%)	*R*
Hartmann (1998)	0.25	5	11.89	4	–	–	36.74	5
Hartmann (2002)	0.22	3	11.70	2	–	–	35.39	2
Nonobe and Ibaraki (2002)	–	–	–	–	–	–	35.86	3
Alcaraz and Maroto (2001)	0.12	2	11.86	3	–	–	36.57	4
Bouleimen and Lecocq (2003)	0.23	4	11.90	5	–	–	37.68	6
Our procedure	0.11	1	11.10	1	10.59	1	33.10	1

by Valls et al., 2003). Because the results for the different algorithms have been obtained using different computers, a direct comparison is not possible. Rather, we will show that our algorithm is able to outperform these heuristics with a specific limit on the number of schedules generated. As measures of algorithmic effectiveness and efficiency, we report the sum of the project makespans, the average deviations from the critical path (except for J30, where we report the average deviation from the optimal solution) and average and maximum CPU times, where available.

Nonobe and Ibaraki (2002) developed a tabu search algorithm for the RCPSP, for which new computational results are reported by Valls et al. (2003). Table 6 shows that we are able to obtain better results using only 5000 schedules, except for J30, where we need 50,000 schedules. Additionally, we require far less computation time, even if we take into account the difference in computer system (Sun Ultra 2 running at 300 MHz versus 1.8 GHz PC).

Recently, Fleszar and Hindi (2004) have developed a heuristic for the RCPSP based on variable

Table 6
Comparative computational results without schedule limit

Author	Problem set									
	J30					J60				
	Sum	Dev. (%)	Avg. CPU	Max. CPU	*R*	Sum	Dev. (%)	Avg. CPU	Max. CPU	*R*
Nonobe and Ibaraki (2002)	28,337	0.06	9.07	–	5	38,697	11.55	26.49	–	8
Fleszar and Hindi (2004)	–	–	0.64	5.86		–	10.94	8.89	80.70	4
Palpant et al. (2004)	–	0.02	22.23	211.00	3	–	10.93	58.04	343.00	3
Valls et al. (2003)	28,335	0.06	1.61	6.15	4	38,671	11.45	2.76	14.61	7
Valls et al. (2004)	28,361	0.13	0.38	1.54	7	38,512	10.98	1.14	7.03	5
Our [5000]	28,355	0.11	0.06	0.12	6	38,554	11.10	0.18	0.27	6
Our [50,000]	28,319	0.01	0.69	1.27	1	38,420	10.71	1.88	2.64	2
Our [500,000]	28,319	0.01	7.16	9.89	1	38,357	10.53	19.61	33.14	1
	J90					J120				
Nonobe and Ibaraki (2002)	46,294	11.25	181.41	–	7	76,600	34.99	645.33	–	7
Fleszar and Hindi (2004)		–	32.43	247.91		–	33.10	219.86	1126.97	4
Palpant et al. (2004)	–	10.54	93.91	508.00	4	–	33.16	318.33	852.00	5
Valls et al. (2003)	46,247	11.12	4.63	25.49	6	76,356	34.53	17.00	43.94	6
Valls et al. (2004)	45,967	10.44	2.53	17.57	3	75,009	32.18	14.52	60.80	3
Our [5000]	46,030	10.59	0.37	0.70	5	75,537	33.10	0.65	0.93	4
Our [50,000]	45,822	10.09	4.03	8.02	2	74,6717	31.57	6.66	9.22	2
Our [500,000]	45,700	9.80	38.87	54.94	1	74,055	30.48	69.57	98.45	1

neighbourhood search. They report good computational results, but requiring substantial computational effort. For sets J60 and J120, Fleszar and Hindi (2004) report average deviations from the critical-path lower bound of 10.94% and 33.10%. Our algorithm is capable of producing deviations of only 10.53% and 30.48% with 500,000 schedules, and 10.71% and 31.57% with 50,000 schedules, respectively. This indicates that we outperform their results, even with a maximum of 50,000 schedules, whereas Fleszar and Hindi (2004) do not set a limit on the number of schedules, which runs to a maximum of more than 1 million for J60 and more than 10 million for J120. They also report high computation times up to a maximum of 1127 seconds (1 GHz processor), compared to a maximum of less than 10 seconds for our procedure (with 50,000 schedules on a 1.8 GHz processor).

In another recent paper, Palpant et al. (2004) present a large neighbourhood search approach in which blocks of activities scheduled within specific time slices are rescheduled by a commercial solver. We achieve lower percentage deviations from the critical-path lower bound with a limit of 50,000 schedules, and with only 5000 schedules for J120. Moreover, the algorithm of Palpant et al. (2004) requires high computation times (on a 4 processor HP 9000 workstation running at 440 MHz), namely between 23 and 489 times as much as our algorithm needs to produce even better results.

Valls et al. (2003) present a heuristic based on critical activity reordering. Although their results for the J30 set are good, and require a 50,000 schedule-limit for our procedure to be able to outperform it, the results are rather disappointing for sets J60, J90 and J120, where our algorithm can produce better results with only 5000 schedules. The CPU time required by Valls et al. (2003) is limited, but our procedure requires even less time, more than offsetting the difference in processor speeds (400 MHz versus 1.8 GHz). This is especially clear for set J120, where Valls et al. (2003) require more than 25 times the CPU time we need to outperform them.

Valls et al. (2004) report excellent results, especially for sets J60, J90 and J120, as shown in Table 6. They show that their results outperform all other state-of-the-art heuristics, although their procedure is not subjected to a schedule limit, contrary to the other procedures. Nevertheless, the authors also demonstrate that even with extended time limits, the quality of the solutions produced by the other heuristics is lower than theirs. With our new procedure, we are able to do even better, using 5000 schedules for J30 and 50,000 schedules for J60, J90 and J120. Note, however, that in order to outperform the results of Valls et al. (2004), our procedure requires more CPU time if we take into account the difference in processor speed (400 MHz versus 1.8 GHz). On average, our procedure requires more time for sets J60, J90 and J120, but we are able to report better solutions for all problem sets.

From these experiments, we can conclude that the proposed heuristic outperforms all existing heuristic algorithms presented in the literature in terms of both solution quality as well as time required, except for Valls et al. (2004), where we do produce better solutions, but at the expense of slightly more required time.

8. Conclusions

In this paper, we have presented a new heuristic procedure for solving the resource-constrained project scheduling problem (RCPSP), one of the most challenging combinatorial optimisation problems in scheduling. The procedure is a population-based evolutionary method that combines elements from scatter search and from a novel method originally introduced for optimising unconstrained continuous functions based on an analogy with electromagnetism theory. We have explained how this electromagnetism heuristic can be extended for application to combinatorial optimisation problems and to the RCPSP, and how it can be integrated into a scatter search framework. The computational results show that the procedure outperforms other state-of-the-art heuristics in the literature, and that it is competitive with the procedure of Valls et al. (2004), which is probably the most effective heuristic presented in the literature to date.

References

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

Birbil, S.I., Fang, S.C., 2003. An electromagnetism-like mechanism for global optimization. Journal of Global Optimization 25, 263–282.

Birbil, S.I., Fang, S.C., Sheu, R.-L., forthcoming. On the convergence of a population-based global optimization algorithm. Journal of Global Optimization.

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

Bouleimen, K., Lecocq, H., 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149, 268–281.

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

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.

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

Demeulemeester, E., Herroelen, W., 1997. New benchmark results for the resource-constrained project scheduling problem. Management Science 43, 1485–1492.

Demeulemeester, E., Vanhoucke, M., Herroelen, W., 2003. A random generator for activity-on-the-node networks. Journal of Scheduling 6, 13–34.

Fleszar, K., Hindi, K.S., 2004. Solving the resource-constrained project scheduling problem by a variable neighbourhood search. European Journal of Operational Research 155 (2), 402–413.

Glover, F., Laguna, M., Martí, R., 2000. Fundamentals of scatter search and path relinking. Control and Cybernetics 39, 653–684.

Glover, F., Laguna, M., Martí, R., forthcoming. Scatter search. In: Ghosh, A., Tsutsui, S. (Eds.), Theory and Applications of Evolutionary Computation: Recent Trends, Springer-Verlag, New York.

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

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

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

Herroelen, W., Demeulemeester, E., De Reyck, B., 1998a. A classification scheme for project scheduling. In: Weglarz, J. (Ed.), Project Scheduling—Recent Models, Algorithms and Applications, International Series in Operations Research and Management Science, vol. 14. Kluwer Academic Publishers, Dordrecht, pp. 77–106 (Chapter 1).

Herroelen, W., De Reyck, B., Demeulemeester, E., 1998b. 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 Productions Management 13 (11), 80–91.

Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 43, 23–40.

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, pp. 147–178.

Kolisch, R., Padman, R., 2001. An integrated survey of deterministic project scheduling. Omega 49 (3), 249–272.

Kolisch, R., Sprecher, A., 1997. PSPLIB—A project scheduling library. European Journal of Operational Research 96, 205–216.

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

Martí, R., Laguna, M., Glover, F., 2006. Principles of scatter search. European Journal of Operational Research, 169, 359–372.

Mingozzi, A., Maniezzo, V., Ricciardelli, S., Bianco, L., 1998. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. Management Science 44, 715–729.

Nonobe, K., Ibaraki, T., 2002. 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, Boston, pp. 557–588.

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

Ozdamar, 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.

Palpant, M., Artigues, C., Michelon, P., 2004. LSSPER: The resource-constraint project scheduling problem with large neighbourhood search. Annals of Operations Research 131, 237–257.

Sprecher, A., 2000. Scheduling resource-constrained projects competitively at modest resource requirements. Management Science 46, 710–723.

Taillard, E.D., Gambardella, L.M., Gendreau, M., Potvin, J.-Y., 2001. Adaptive memory programming: A unified view of metaheuristics. European Journal of Operational Research 134, 1–16.

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

Valls, V., Ballestín, F., Quintanilla, S., 2004. A population-based approach to the resource-constrained project scheduling problem. Annals of Operations Research 131, 305–324.