Annals of Operations Research 131, 237–257, 2004

LSSPER: Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search

MIREILLE PALPANT, CHRISTIAN ARTIGUES and PHILIPPE MICHELON
Laboratoire d'Informatique d'Avignon, 339, chemin des meinajariès, Agroparc, BP 1228, 84911 Avignon Cedex 9, France

Abstract. This paper presents the Local Search with SubProblem Exact Resolution (LSSPER) method based on large neighbourhood search for solving the resource-constrained project scheduling problem (RCPSP). At each step of the method, a subpart of the current solution is fixed while the other part defines a subproblem solved externally by a heuristic or an exact solution approach (using either constraint programming techniques or mathematical programming techniques). Hence, the method can be seen as a hybrid scheme. The key point of the method deals with the choice of the subproblem to be optimized. In this paper, we investigate the application of the method to the RCPSP. Several strategies for generating the subproblem are proposed. In order to evaluate these strategies, and, also, to compare the whole method with current state-of-the-art heuristics, extensive numerical experiments have been performed. The proposed method appears to be very efficient.

Keywords: resource-constrained project scheduling problem, large neighbourhood search

1. Introduction

The problem of searching a given neighbourhood of an NP-hard problem can be itself defined as a combinatorial optimization problem. In this paper we consider the neighbourhood of a solution to an n variables NP-hard problem P as the set of solutions to a p variables subproblem obtained directly from P by fixing $n - p$ variables to their current value. With no restrictive assumption on p, finding the best neighbour is likely to be itself an NP-hard problem and the number of neighbours is exponential in p.

When p is small enough, an exhaustive enumeration of the neighbours can be performed but the so-defined neighbourhood is unlikely to contain interesting feasible solutions. In this paper we consider larger values of p, presumably defining a (very) large neighbourhood, and we search for the best neighbour with an implicit enumeration technique or a heuristic.

Successful applications of large neighbourhood search (LNS) are available in the literature for various problems. For the quadratic assignment problem (QAP), the MIMAUSA method designed by Mautor and Michelon (1997) builds at each iteration a reduced QAP and solves it by branch and bound. For the vehicle routing problem (VRP), Shaw (1998), Gendreau, Pesant, and Rousseau (2002), and recently Bent and Hentenryck (2001), consider the removal of several customer visits and re-insert them by using limited discrepancy search (LDS). The latter succeeds in improving best published solutions of standard VRP instances with time windows. In (Taillard and Voss, 2002),

Taillard and Voss propose a general algorithm (POPMUSIC) in which subproblems are built from subparts of the solution. A survey of very large scale neighbourhood search techniques can be found in (Ahuja et al., 2002).

In this paper, we consider the resource-constrained project scheduling problem (RCPSP) in which a set of activities has to be scheduled, subject to precedence constraints and resource limitations with the objective to minimise the total project duration or makespan. This problem has received special attention in the last decades because of its relative generality and its numerous practical applications (see (Brucker et al., 1999; Kolisch and Padman, 2001) for recent surveys). Furthermore it is particularly hard to attack. Thus, the best exact methods proposed so far are unable to solve some benchmark instances consisting of 60 activities and 4 resources.

The use of LNS for solving scheduling problems is not new. Indeed, the shifting bottleneck heuristic, initially designed by Adams, Balas, and Zawack (1988), is one of the most popular heuristics to solve the famous job-shop problem (JSP) and some of its extensions including multiresource operations, multipurpose machines, setup times and deadlines (see for instance (Balas et al., 1998; Schutten, 1998)). In the shifting bottleneck heuristic, the neighbourhood is defined by deriving a one-machine subproblem from the current solution. The best neighbour is then found by solving this problem exactly. For the classical JSP, other LNS heuristics using various subproblem generation and resolution schemes have been proposed in (Applegate and Cook, 1991; Baptiste, Le Pape, and Nuijten, 1995; Caseau and Laburthe, 1999). The latter approach is the forget-and-extend heuristic of Caseau and Laburthe which has obtained excellent results on hard job-shop instances. The neighbourhood is defined by keeping the processing order of operations, either belonging to a variable set of resources (as for the shuffling procedure of Applegate and Cook (1991)), or scheduled in a variable temporal slice. Hence different neighbourhood types are considered for diversification purposes. The corresponding subproblems are solved by LDS combined with the powerful constraint propagation "shaving" technique. Some LNS methods have also been proposed recently for other scheduling problems such as the single machine total weighed tardiness (Congram, Potts, and Van de Velde, 2002), parallel machines (Frangioni, Scutellà, and Necciari, 2004) and unrelated machines (Sourd, 2001) problems.

Despite the success of LNS methods for (job-shop) scheduling, to the best of our knowledge no heuristic of this type for the RCPSP has been proposed yet though it is an extension of the JSP. To fill this gap, we propose in this paper a new LNS method for the RCPSP. This method differs from the ones cited above by the way it explores the neighbourhood. Indeed, we concentrate our efforts on the generation of the subproblem, leaving its resolution to a commercial solver. The size of the subproblem self-adapts to the time spent by the solver to provide the solution. These features give our method a practical interest since it is designed with almost no assumption on the method used to explore the neighbourhood.

In section 2, we briefly describe the principles of the used LNS method in a general context. In section 3, we propose a solution approach for the RCPSP based on the LNS

method. In section 4, we conclude by discussing the computational results obtained on standard benchmark instances.

2. General description of the method

Let us consider the resolution of a general optimisation problem

$$P\colon \min_{X\in\mathcal{X}} f(X), \quad \text{with } X = (x_1, \ldots, x_n).$$

Throughout the paper, a capital letter (e.g., X) denotes a vector of decision variables whereas an overline capital letter (e.g., $\overline{X}$) represents a vector of values of the corresponding variables.

The proposed LNS metaheuristic is widely inspired by the MIMAUSA method of Mautor and Michelon (1997, 2001). It aims at alternating intensification and diversification phases (Glover and Laguna, 1997) in the search of solutions.

Intensification phases consist of exploring deeply a given subset of the feasible region $\mathcal{X}$ by solving successive subproblems. Thus, at each iteration s, a subproblem Π^s of size p is generated by fixing $n - p$ variables to the value they have in current solution $\overline{X}^{s-1}$ to P. Solving Π^s generates a solution $\overline{Y}^s$ whose extension to global problem P provides the neighbour solution $\overline{X}^s$ of $\overline{X}^{s-1}$.

Conversely, diversification aims at visiting a subset of the feasible region $\mathcal{X}$ which has not been explored yet. It consists in applying a diversification operator to the current solution.

The method, which returns final solution X^*, is detailed in figure 1.

Step 1 is assumed to compute (quickly) a feasible solution $\overline{X}^0$ to P using any problem-dependent heuristic. The other major points of the algorithm are developed in the subsequent subsections:

- the subproblem generation and the auto-adjustment of its size are described in section 2.1,
- subproblem resolution strategies are discussed in section 2.2,
- the generation of neighbours and additional diversification procedures are presented in section 2.3.

2.1. A self-adapting subproblem construction

At each iteration s, a subproblem of varying size p

$$\Pi^s\colon \min_{Y\in\mathcal{Y}} g(Y), \quad \text{with } Y = (y_1, \ldots, y_p),$$

is constructed by using problem P and its current solution $\overline{X}^{s-1}$. Roughly, the construction method consists in selecting $p \leqslant n$ variables $x_{i_1}, \ldots, x_{i_p}$ (renamed $y_1, \ldots, y_p$) of

Begin
1. Compute an initial solution $\overline{X}^0$ to problem P
2. Set $\overline{X}^* := \overline{X}^0$, $s := 1$
3. Initialise p
4. **Repeat**
5. Generate a subproblem Π^s of size p using P and its current solution $\overline{X}^{s-1}$
6. Solve Π^s within a maximum time limit H and store actual resolution time t^s
7. **If** a solution $\overline{Y}^s$ to Π^s is obtained **then**
8. Compute new solution $\overline{X}^s$ by extending $\overline{Y}^s$
9. **Else**
10. Set $\overline{X}^s := \overline{X}^{s-1}$
11. **End If**
12. **If** $f(\overline{X}^s) < f(\overline{X}^*)$ **then**
13. Set $\overline{X}^* := \overline{X}^s$
14. **End If**
15. **If** the diversification condition is verified **then**
16. Modify $\overline{X}^s$ by a diversification operator
17. **End If**
18. Adjust p by using statistics on $t^s, \ldots, t^{s-q}$
19. $s := s + 1$
20. **Until** stop criterion is met

End

Figure 1. General algorithm of the proposed method.

P and in replacing in the constraints of P the $n - p$ remaining variables $x_{i_{p+1}}, \ldots, x_{i_n}$ by their current values $\overline{x}_{i_{p+1}}, \ldots, \overline{x}_{i_n}$.

Any feasible solution $\overline{Y}$ of subproblem Π^s has to be such that $\overline{Z}$ verifying $\overline{z}_{i_1} = \overline{y}_1, \ldots, \overline{z}_{i_p} = \overline{y}_p$ and $\overline{z}_{i_{p+1}} = \overline{x}_{i_{p+1}}, \ldots, \overline{z}_{i_n} = \overline{x}_{i_n}$ is a feasible solution of P. Let $\overline{Y}^*$ denote the optimal solution of Π^s and let $\overline{Z}^*$ denote the corresponding extension to P. Objective function g is such that $f(\overline{Z}^*) \leqslant f(\overline{X})$.

A stronger alternative consists of considering additional constraints such that for any feasible solution $\overline{Y}$ of subproblem Π^s, $f(\overline{Z}) \leqslant f(\overline{X})$. In such a case, any feasible solution of subproblem Π^s can be obviously extended to obtain a neighbour $\overline{X}^s$ of $\overline{X}^{s-1}$ for problem P verifying $f(\overline{X}^s) \leqslant f(\overline{X}^{s-1})$.

The auto-adjustment of size p is related to statistics on q consecutive values of $t^s, \ldots, t^{s-q}$, $q \geqslant 1$, where t^s denotes the time spent at iteration s to solve subproblem Π^s. If t^s tends to increase, then p is decreased, until a lower bound $\underline{p}$ is reached. If t^s tends to decrease, the size of the subproblem is increased, until $p = n$.

Once p is determined, the selection of the p variables results from a problem-dependent analysis of the constraints of P with the objective to ensure that:

- the neighbourhood defined by Π^s has a "reasonable" size, i.e. contains a large number of solutions that can be explored by an implicit enumeration method or a heuristic,
- the neighbourhood defined by Π^s contains promising solutions,
- the feasible region $\mathcal{X}$ of P is sufficiently explored by solving consecutive subproblems Π^q, $q \geqslant 1$ (diversification of the search): the p variables must be selected in a non-uniform way.

2.2. Resolution of the subproblem and intensification of the search

Once defined, an attempt to solve the subproblem Π^s is performed within a maximal allotted time H. We assume the resolution method to be any (external to our method) exact or approximate solution approach for problem Π^s. If feasible (possibly optimal) solution $\overline{Y}^s$ to Π^s is found then the time $t^s \leqslant H$ spent to obtain it is stored. Otherwise, t^s is set to an arbitrary large value. A large time limit H corresponds to an intensification of the search since more time is spent in the search of the best solution of Π^s, i.e. the best neighbour, and is more likely coupled with the use of an exact method. Conversely, setting H to a small value aims at finding (hopefully) quickly a feasible solution of Π^s.

2.3. Generation of the neighbour solution and additional diversification of the search

Once a solution $\overline{Y}^s$ to problem Π^s has been obtained, solution $\overline{X}^s$ has to be generated by extending $\overline{Y}^s$ to problem P. The simplest way consists in replacing in $\overline{X}^{s-1}$ values $\overline{x}_{i_1}, \ldots, \overline{x}_{i_p}$ by values $\overline{y}_1, \ldots, \overline{y}_p$. In practice, some additional improvements can be performed depending on the problem structure.

Although the varying size of the subproblem and its non-uniform generation scheme bring some diversification to the search, additional diversification procedures can be used to explore other solution subspaces. These can consist of increasing considerably the subproblem size, accepting non improving neighbours, rebuilding new solutions from scratch, etc.

3. Application to the resource-constrained project scheduling problem

In this section, we adapt the method presented in section 2 for the resource-constrained project scheduling problem. A formulation of this problem is given in section 3.1. In section 3.2, we give a brief overview of schedule generation schemes, heuristics and metaheuristics proposed in the literature for the RCPSP. Then, we detail the implementation of the proposed method. The initial solution generation method is given in section 3.3. The subproblem generation and resolution procedures are presented in sections 3.4 and 3.5, respectively. Finally, the neighbour generation and the additional diversification procedures are described in section 3.6.

3.1. Problem description

In the resource-constrained project scheduling problem, $A = \{1, \ldots, n\}$ denotes the set of activities and $R = \{1, \ldots, m\}$ denotes the set of resources. Each resource $k \in R$ has a limited availability R_k. Each activity $i \in A$ has a duration $p_i \geqslant 0$ and a request $r_{ik} \geqslant 0$ on each resource $k \in R$. The activities are organised in a project represented by an activity-on-node graph (also called project network) $G = \{V, E\}$, where $V = A \cup \{0, n+1\}$ is the set of nodes representing the activities and E is the set of directed arcs representing the precedence constraints. A directed arc $(i, j) \in E$ means that activity i must be completed when activity j starts. 0 and $n+1$ are dummy activities representing the start and the end of the schedule, respectively. Their processing times and requests are all set to 0. Node 0 is connected to any activity without predecessor and node $n+1$ is connected to any activity without successor. The variables of the problem being the starting times $S = (S_1, \ldots, S_{n+1})$, the RCPSP can be formulated as follows:

$$(P) \quad \min S_{n+1} \tag{1}$$

$$S_j \geqslant S_i + p_i \qquad \forall (i, j) \in E, \tag{2}$$

$$\sum_{i \in \mathcal{A}(\tau, S)} r_{ik} \leqslant R_k \quad \forall k \in R, \forall \tau \in \{0, \ldots, \mathrm{UB}\} \tag{3}$$

where $S_0 = 0$ and $\mathcal{A}(\tau, S)$ is the set of activities in process at time τ in solution S, i.e. verifying $S_i \leqslant \tau < S_i + p_i$.

This problem is strongly NP-hard, as an extension of the job-shop problem. Figure 2 displays an example taken from (Klein and Scholl, 1999) with 10 non-dummy activities and a single resource of 4 units. In part (a) of the figure, the project network is displayed. Under each node, the duration and the request of the corresponding activity are given, respectively. In part (b) of the figure, a feasible schedule of makespan 24 is represented by way of an extended Gantt chart.

3.2. Schedule generation schemes, metaheuristics and neighbourhoods for the RCPSP

In this section we first give a short overview of the constructive heuristics designed for the RCPSP. Then, we discuss the different neighbourhoods that have been used so far for this problem. For a recent state-of-the-art review of heuristics for the RCPSP we refer to (Hartmann and Kolisch, 2000).

The simplest heuristics are constructive. They build a feasible solution starting from scratch. A constructive heuristic has generally n steps, where n denotes the number of activities. At each step, a single activity is selected for being scheduled among a set of candidate activities. The way this set is generated and the scheduling process of the selected activity are both determined by a schedule generation scheme (SGS). A priority rule generally determines which activity is selected for being scheduled among the candidate ones.

Two SGS are commonly used in the literature: the parallel SGS and the serial SGS. The serial SGS, used in the present study, puts in the set of candidate activities

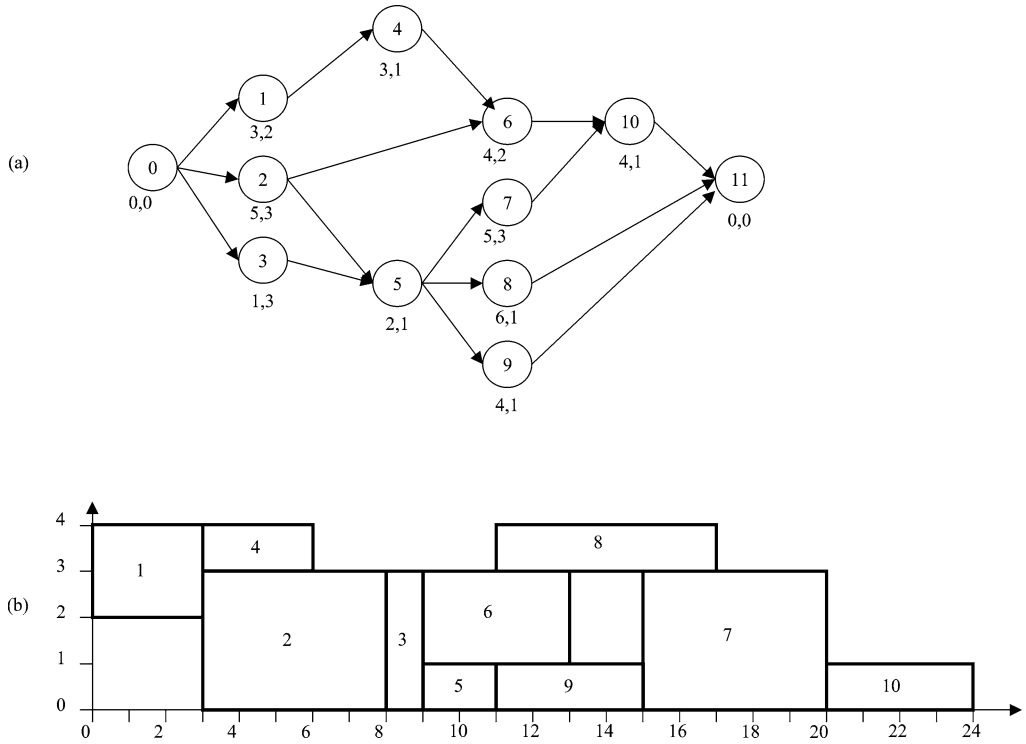

Figure 2. A RCPSP example (Klein and Scholl, 1999) and a feasible solution.

any activity not already scheduled that has either no predecessor or all its predecessors already scheduled. The candidate activity selected with the priority rule is scheduled at its earliest precedence-feasible and resource-feasible starting time, taking the already scheduled activities into account. The set of all schedules that can be generated with the serial SGS is called the set of active schedules and is dominant w.r.t. makespan minimisation. We refer to (Kolisch and Hartmann, 1999) for a more precise description of the serial and parallel SGS.

We roughly divide the heuristics proposed for the RCPSP in two categories. The first category consists in the single and multi-pass priority-rule based methods. A single pass priority rule-based method performs a single call to the serial or the parallel SGS with a priority rule. Multi-pass priority-rule based methods perform several calls of a given SGS. Each time, a different priority-rule is used, possibly by biasing the selection through a random device. We refer to (Hartmann and Kolisch, 2000) and (Schirmer, 2000) for details on multi-pass random-biased priority-rule based methods. Another variant of a multi-pass method is the iterative forward/backward scheduling method (Li and Willis, 1992). We describe this method with more details in section 3.3 since we have used it for both building an initial solution and improving the current solution.

The second category includes the metaheuristics and the neighbourhood search methods. Most of these methods are based on the serial SGS or on a variant named

the "list scheduling algorithm" (Kolisch and Hartmann, 1999; Hartmann and Kolisch, 2000). Thanks to the dominance property of the set of active schedules, there exists an input priority vector (or an input list L^* for the variant) of the serial SGS leading to the optimal schedule. Most of the metaheuristics proposed for the RCPSP (some of them being evaluated in (Hartmann and Kolisch, 2000)) take advantage of this property and restrict the search to priority vectors or to feasible lists. This is for example the case of simulated annealing approaches (Lee and Kim, 1996; Cho and Kim, 1997; Bouleimen and Lecocq, 2003), genetic algorithms (Leon and Ramamoorthy, 1995; Lee and Kim, 1996; Hartmann, 1998; Kohlmorgen, Schmeck, and Haase, 1999), tabu search (Pinson, Prins, and Rullier, 1994; Baar, Brucker, and Knust, 1998; Valls, Ballestin, and Quintanilla, 2000; Nonobe and Ibaraki, 1999; Thomas and Salhi, 1998), best fit search (Naphade, Wu, and Storer, 1997), ant colony algorithm (Merkle, Middendorf, and Schmeck, 2002). In these approaches, a move computes a neighbour activity list or a neighbour priority vector. However, it has to be pointed out that for the activity list as well as for the priority vector representation, the number of neighbour activity lists (or priority vectors) does not necessary reflect the number of actual neighbour solutions in terms of activity starting times. Indeed, it is well known that several different lists or priority vectors can lead to the same solution.

Other neighbourhood search methods based on different schedule representations have been proposed, like the shift vector representation (Sampson and Weiss, 1993), the schedule scheme representation (Baar, Brucker, and Knust, 1998), the disjunctive arc based representation (Bell and Han, 1991) and the activity-on-node/network flow representation (Artigues, Michelon, and Reusser, 2003).

In contrast with these approaches, we focus in this paper on the direct representation of the schedule by the vector of starting times. Indeed, as it will be presented in the next subsections, this allows to define the search for the best neighbour as a scheduling subproblem. In this framework, an interesting approach has been used in (Mausser and Lawrence, 1997), where the makespan of a given solution is improved by rescheduling entirely at each step a block of activities. A block is composed by the activities entirely scheduled in a given time slice, as in the Forget and Extend heuristic (Caseau and Laburthe, 1999). The rescheduling of a given block potentially represents a large neighbourhood. Hence, such a block structure is considered in the present study.

3.3. Generation of an initial solution

In this subsection, we describe how we generate an initial solution $\overline{S}^0$ for our implementation of LNS for the RCPSP. $\overline{S}^0$ is generated with a variant of the iterative forward/backward heuristic (Li and Willis, 1992). This heuristic alternates forward and backward passes of the serial SGS, computing each time a feasible schedule. The forward pass computes a solution by applying the serial SGS to the RCPSP problem P, whereas the backward pass computes a solution by considering its "mirror" problem MP obtained by reversing all arcs of the project network. In our variant, we ensure that each generated schedule has a makespan non-greater than the previous one, thanks to

the selected priority rules. This property together with the stop condition ensures a fast convergence.

In the first forward pass, we generate a schedule with the serial SGS and the MINLFT priority rule.[1] Then we apply iteratively backward and forward passes under the following scheme. Let $\overline{FS}$ ($\overline{FF}$) be the start (finish) time vector computed by the forward pass. We set $\overline{S}^0 := \overline{FS}$. The backward pass consists in applying the serial SGS to *MP*, by taking as priority vector, $\overline{FF}_{n+1} - \overline{FF}$. Let $\overline{BS}$ ($\overline{BF}$) be the start (finish) time vector computed by the backward pass. It can be easily shown that we can set $\overline{S}^0 := \overline{BF - \overline{BF}_0}$ without increasing the makespan. Indeed, such a backward pass amounts to applying a right shift to each activity in the solution $\overline{FS}$, in the non increasing order of the completion times $\overline{FF}$, the right shift being bounded from above by the current makespan.

If the makespan has not been decreased, the process stops. Otherwise, the next forward pass consists in applying the serial SGS to P, by taking the new current start time vector $\overline{S}^0$ as priority vector. Again, the obtained start time vector $\overline{FS}$ verifies $\overline{FS}_{n+1} \leqslant \overline{S}^0$. The process iterates until two consecutive forward and backward passes obtain the same makespan.

3.4. Generation of the subproblem

In contrast with most of the state-of-the-art methods, we do not use any special solution representation such as activity lists, priority vectors or disjunctive graphs. Hence a solution is only represented in terms of a precedence- and resource-feasible start time vector $\overline{S}$.

At each iteration $s \geqslant 1$ of the method, the subproblem Π^s and, consequently, the neighbourhood are defined by a set $A^s \subseteq A$ of p activities $A^s = \{i_1, \ldots, i_p\}$ excluding dummy activities 0 and $n+1$. Once these p activities have been selected, the subproblem Π^s can be stated as follows. Each activity $j \in A \setminus A^s$ being "frozen" at its current start time value $\overline{S}_j^{s-1}$, find a feasible schedule minimising the makespan of the remaining activities.

In practice, freezing the start time of some activities is equivalent to generate a new RCPSP applying the 3 following steps:

- We build a reduced project network $G^s = (A^s, E^s)$ including only the non-frozen activities where $E^s = \{(i, j) \mid i, j \in A^s, (i, j) \in E\}$.
- We modify the resource set R such that the availability $R_{k\tau}$ of each resource $k \in R$ at time period τ reflects the possibly frozen activities in process during τ:

$$R_{k\tau} = R_k - \sum_{j \in (A \setminus A^s) \cap \mathcal{A}(\tau, \overline{S}^{s-1})} r_{jk}.$$

- For each activity $i \in A^s$, we compute recursively a time window $[ES_i, LS_i]$ by taking the predecessor(s) and successor(s) of i as well as resource availability into account.

First, we compute earliest and latest start times taking into account precedence constraints of both frozen and selected activities:

$$ES_i^{pred} = \max\Big\{\max_{j\in A\setminus A^s,(j,i)\in E} \overline{S}_j^{s-1} + p_j, \max_{j\in A^s,(j,i)\in E^s} ES_j + p_j\Big\},$$
$$LS_i^{succ} = \min\Big\{\min_{j\in A\setminus A^s,(i,j)\in E} \overline{S}_j^{s-1} - p_i, \min_{j\in A^s,(i,j)\in E^s} LS_j - p_i\Big\}.$$

As it can be seen in the computation of LS_i^{succ}, we also force any activity to be completed at the current makespan value $\overline{S}_{n+1}^{s-1}$.

Second, we right (resp. left) shift the earliest (resp. latest) start time until the activity can be processed without interruption w.r.t. resource availability:

$$ES_i = \min\big\{t \mid t \geqslant ES_i^{pred}, \forall k \in R, \forall \tau = t, \ldots, t + p_i - 1, R_{k\tau} \geqslant r_{ik}\big\},$$
$$LS_i = \max\big\{t \mid t \leqslant LS_i^{succ}, \forall k \in R, \forall \tau = t, \ldots, t + p_i - 1, R_{k\tau} \geqslant r_{ik}\big\}.$$

The variables of the subproblem built at iteration s being the starting times $(S_{i_1}^s, \ldots, S_{i_p}^s)$, Π^s can be formulated as follows:

$$(\Pi^s) \quad \min_{i\in A^s}\{\max S_i^s\} \tag{4}$$
$$S_i^s \geqslant ES_i \qquad \forall i \in A^s, \tag{5}$$
$$S_i^s \leqslant LS_i \qquad \forall i \in A^s, \tag{6}$$
$$S_j^s \geqslant S_i^s + p_i \qquad \forall (i, j) \in E^s, \tag{7}$$
$$\sum_{i\in A^s\cap\mathcal{A}(\tau,S^s)} r_{ik} \leqslant R_{k\tau} \quad \forall k \in R, \forall \tau \in \big\{0, \ldots, \overline{S}_{n+1}^{s-1}\big\}. \tag{8}$$

The subproblem Π^s can be defined as a RCPSP with time windows and varying resource availability, which makes it in a sense more general than the original problem! It is also easy to verify that any feasible solution of the subproblem can lead to a feasible new solution $\overline{S}^s$ of (P) by setting $\overline{S}_i^s = \overline{S}_i^{s-1}\ \forall i \in A \setminus A^s$. However, $\overline{S}_{n+1}^s$ may be different from objective function value (4). We will see in section 3.6 how neighbour solution $\overline{S}^s$ is further improved.

In figure 3, the subproblem generation is applied on the current solution of the illustrative example displayed in figure 2, assuming that p is set to 4 and that activities $A^s = \{7, 8, 9, 10\}$ have been selected. In part (a) of the figure, the selected activities are displayed. Part (b) of the figure displays the computed time windows. Part (c) of the figure shows the resource profile after removing the units required by the frozen activities. Last, part (d) of the figure displays the project network of the subproblem.

We now explain how to determine the number p of activities constituting the subproblem Π^s and how the p activities are selected.

As mentioned in section 2.1, the computation of p self-adapts in function of the time spent to solve the subproblem at the previous iterations. Let H denote the maximum CPU time allotted to the subproblem resolution method. Let t^{s-1} be the time spent to solve the subproblem at iteration $s-1$. Let $t = \sum_{q=s-1,\ldots,s-5} t^q$ the total time spent

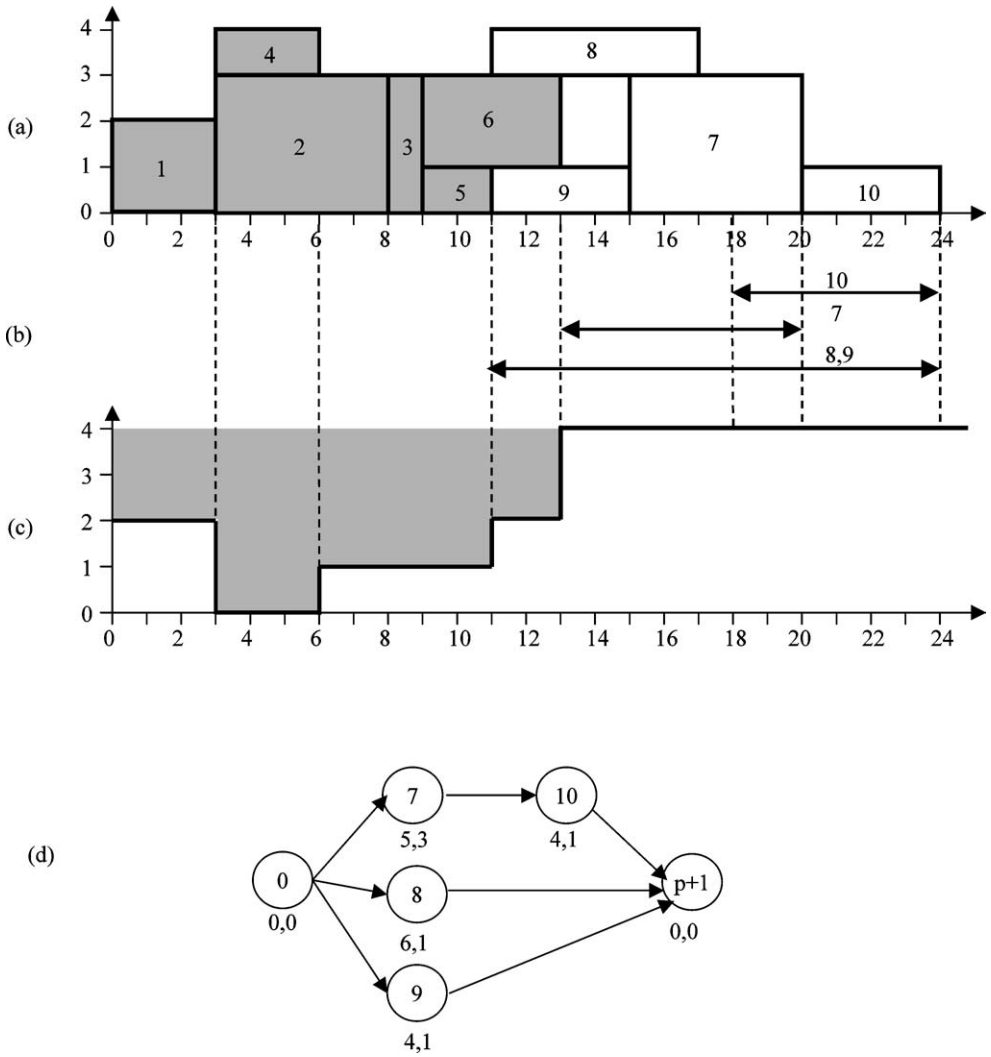

Figure 3. Example of subproblem generation with $p = 4$.

on the subproblem resolution over the last 5 iterations. We have used the following empirical rules. If $t \leqslant H$ then we set $p := p + 2$. If $H < t \leqslant 5H/2$ then we set $p := p + 1$. If $5H/2 < t \leqslant 4H$ then we set $p := p - 1$. If $4H < t \leqslant 5H$ then we set $p := p - 2$. No decrease on p is made if a given lower bound $\underline{p}$ is reached.

To select the p activities, i.e. to build the set A^s, we have compared in our experiments the 5 following possibilities (see section 4):

1. Critical&Random. The p activities are selected randomly but a higher probability is assigned to critical activities. We decide that an activity is critical when it has been scheduled at the same start times on the last two passes of the forward/backward method.

2. Random&Project Predecessors. The p activities are selected randomly but each time an activity is selected, its immediate predecessor(s) in the project network is (are) also selected.
3. Random&Contiguous Predecessors. The p activities are selected randomly but each time an activity i is selected, any activity verifying $\overline{S}_j^{s-1} + p_j = \overline{S}_i^{s-1}$ is selected.
4. Random&All Predecessors. The p activities are selected by combining 2 and 3.
5. Block. A single activity i is randomly selected and included in A^s. Then, each activity j contiguous to i or scheduled in parallel with i, (i.e. verifying $\overline{S}_i^{s-1} - p_j \leqslant \overline{S}_j^{s-1} \leqslant \overline{S}_i + p_i$) is included in A^s in the limit of p activities. If this limit is not reached, we iterate the process from the second, third, ..., activity added to A^s. For diversification purpose, the activities are considered for being included in A^s in a predefined random order.

For selection methods 2–5, the objective is to generate subproblems having sufficient independence with the rest of the solutions to make the represented neighbourhood large enough. Selection scheme 5 can be seen as an adaptation of the methods proposed in (Caseau and Laburthe, 1999; Mausser and Lawrence, 1997). The subproblem displayed in figure 3 is obtained by selection scheme 5, $p = 4$ and a random selection of activity 7. The only 3 activities verifying the condition $\overline{S}_7^{s-1} - p_j \leqslant \overline{S}_j^{s-1} \leqslant \overline{S}_7^{s-1} + p_7$ are 8, 9 and 10. Hence any selection order would have lead to this subset of activities for $p = 4$.

3.5. Solving the subproblem

Our approach consists in evaluating the generality of the proposed method and its applicability in practical situations by leaving the subproblem resolution to a commercial solver. In our experiments, we have used a constraint programming solver specialized in scheduling and an integer linear programming solver. For the CP solver, constraints are directly written since a library of scheduling constraints is available. For the ILP solver, we use an extension of the well-known ILP formulation of Pritsker, Waters, and Wolfe (1969) to time windows and time-varying resource availability. In the model of Pritsker, Waters, and Wolfe, decision variables are the 0–1 variables $x_{i\tau} = 1$ if and only if activity i starts at time τ. In particular, there is a resource constraint per resource and per periods which makes easy the extension of the model to non-constant resource availability. In both cases, no particular tuning is performed (see section 4). The solver is allotted H time units and it returns the best solution found (if any) during this time limit.

In figure 4 we give in part (b) an example of solution $\overline{S}^s$ obtained by a direct extension of the solution returned by the solver (part (a)) for the subproblem of figure 3. In this case, an improved (and optimal) solution of makespan 22 is obtained.

3.6. Neighbour generation and additional diversification

At iteration s, a feasible neighbour solution $\overline{S}^s$ of $\overline{S}^{s-1}$ is directly obtained by extending the subproblem solution (see section 3.4).

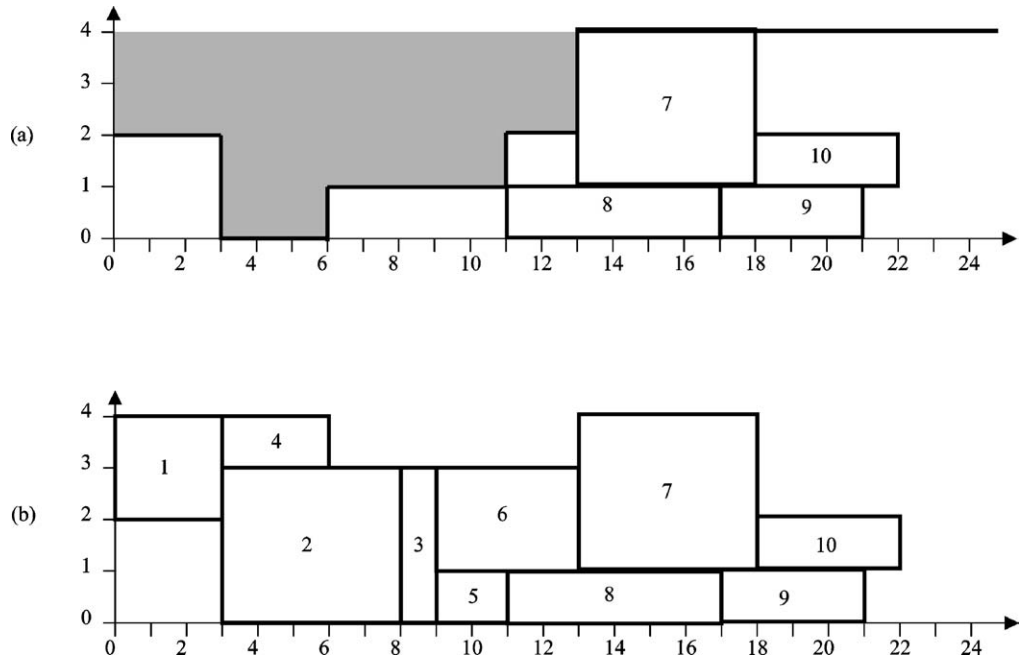

Figure 4. Resolution of the 4 activities subproblem of figure 3.

In the general case, such a direct extension may be easily improved. Indeed, all frozen activities scheduled after the activities of the subproblem are likely to be possibly left shifted, if the subproblem has been optimised positively. Hence, the direct extension serves as the first priority vector of the forward/backward approach presented in section 3.3. We illustrate this process in figure 5. A solution $\overline{S}^{s-1}$ of makespan 25 is displayed in part (a) of the figure. The frozen activities are displayed in grey. Suppose that solving the corresponding subproblem gives the suboptimal solution displayed in part (b) and consider its direct extension for which we obtain a makespan of 25. However, the frozen activities can obviously be left shifted. We apply a first forward pass of the serial SGS by setting the priority vector to the start times of the direct extension. The schedule of makespan 24 displayed in part (c) is obtained. The backward pass is applied (as explained in section 3.3) generating the schedule of makespan 22 displayed in part (d). The next forward pass obtains the same makespan which ends the process and gives the neighbour solution $\overline{S}^{s}$, which is here also optimal.

To perform additional diversification of the search, when the current solution has not been improved during a number of iterations (arbitrarily set to n), a new solution is generated by a call to the forward/backward heuristic with a random priority rule for the first forward pass. This amounts to generating a new starting point for the proposed method.

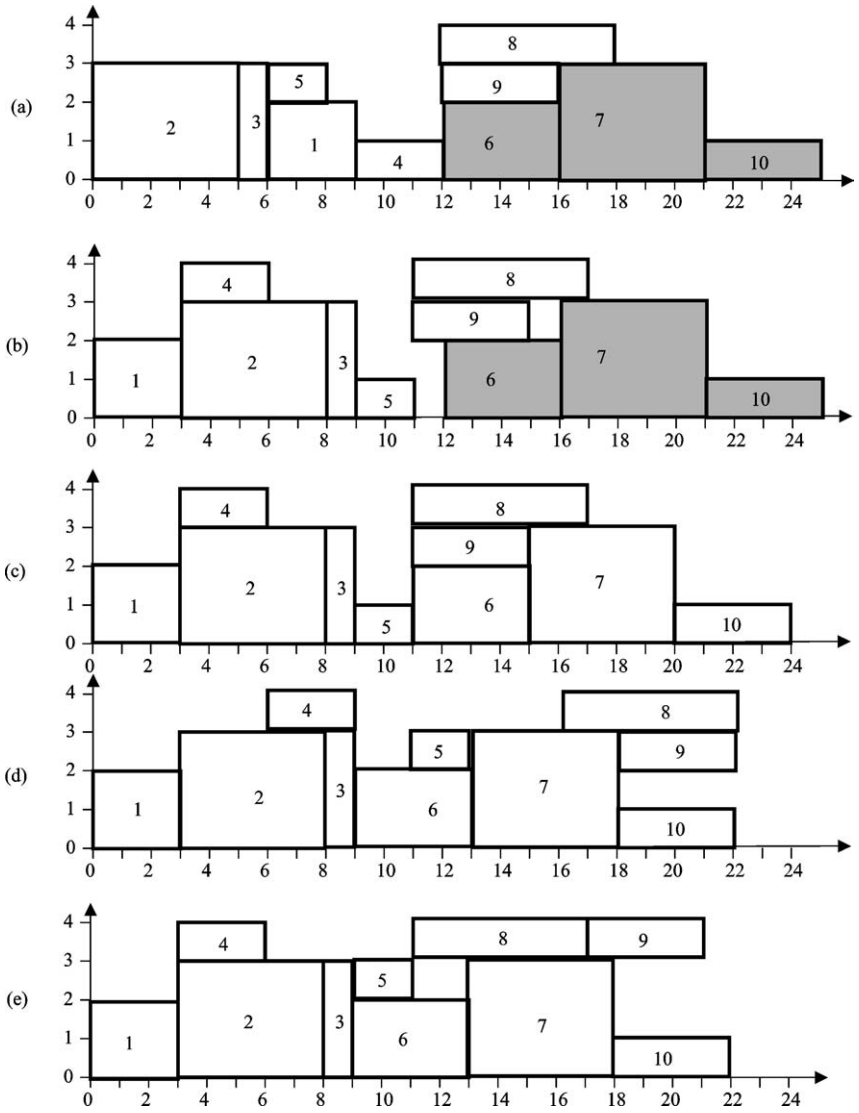

Figure 5. Improvement of the direct extension with the forward/backward heuristic.

4. Computational experiments

We have implemented the proposed LNS heuristic LSSPER in C++. For the resolution of the subproblem, generated by the Block strategy, we use the specialised ILOG SCHEDULER 5.0 constraint programming solver. We set empirically maximum resolution time H to 0.5 s, the maximal total number of iterations to $10 \cdot n$ and initial subproblem size p to 15.

Experiments have been performed on a PC with 1 GB RAM and a 2.3 GHz processor. We have tested our method on the 110 "easy" Patterson instances (from 6 to 51

Table 1
Best results of the proposed LNS Search.

Prob.	av. (max) ΔUB	av. ΔLB	# best	av. (max) CPU	av. (max) # sched
PAT	0 (0)	–	110/110	1.60 (59)	–
ALV	−0.46 (5.19)	46.24	45/48	232.23 (498)	–
BL	0.11 (4.17)	–	38/39	17.61 (66)	–
KSD30	0 (0)	–	480/480	10.26 (123)	830 (1120)
KSD60	0.22 (3.54)	10.81	413/480	38.78 (223)	1622 (2262)
KSD90	0.40 (5.65)	10.29	379/480	61.25 (309)	2441 (3488)
KSD120	1.51 (20.91)	32.41	241/600	207.93 (501)	3396 (5000)

activities) (Patterson, 1984) (PAT), the 39 highly cumulative Baptiste/Le Pape instances with 20 activities (Baptiste and Le Pape, 2000) (BL), the 48 Alvarez instances with 103 activities (Alvarez-Valdés and Tamarit, 1989) (ALV) and the 2040 Kolisch et al. instances with 30, 60, 90 and 120 activities (Kolisch, Sprecher, and Drexl, 1998) (KSD).

Our results are displayed in table 1. For each problem set, we display in the first two columns the average/maximal deviation from the optimum or from the best known upper bound and the average deviation from the critical path lower bound for the problems whose optimal makespans are still unknown. The number of times we obtain the best solution, the average/maximal CPU time required and the average/maximal number of generated schedules are indicated in the next columns. This latter value corresponds to the total number of schedules generated by our method for the resolution of an instance. It sums the number of solutions found by the solver during the subproblem resolution and the number of schedules computed by the forward/backward heuristic.

For all instances, except for KSD 120 set, the solutions found by LSSPER are on average within 0.5% above the best known solutions. At the time of the experiments, we have found new best solutions on the KSD and ALV sets. We still have 14, 9 and 4 best known solutions for the KSD 60, 90 and 120 sets,[2] respectively. We have solved to optimality all the Baptiste/Le Pape instances except one. However we need twice more CPU time to solve these 20 activities instances than for the 30 activities KSD instances. This seems to confirm the hardness of these highly cumulative instances (Baptiste and Le Pape, 2000) whereas some (but not all!) KSD instances are very easy.

To further analyse our results we compare them with current best state-of-the-art heuristics, all limited to 5000 generated schedules, on the KSD instances: the hybrid genetic algorithm of Valls, Ballestin, and Quintanilla (2002), the self-adapting genetic algorithm of Hartmann (1998), the tabu search method of Nonobe and Ibaraki (1999), the simulated annealing of Bouleimen and Lecocq (2003) and the random biased sampling method of Kolish (1996). Table 2 displays for each method the average deviation above the optimum (KSD30) or CPM lower bound (KSD 60, 90, 120).

The results show that our approach is very competitive with the approaches encountered in the literature. The results on the KSD instances are better than the ones of all the methods presented here. As for these approaches, the average number of generated schedules does not exceed 5000. However, this has to be tempered by the possibly

Table 2
Comparison with state-of-the-art approaches.

Method	av. ΔUB (KSD30)	av. ΔLB (KSD60)	av. ΔLB (KSD90)	av. ΔLB (KSD120)
LSSPER	0	10.81	10.29	32.41
Valls et al. (2002)	0.06	11.10	10.46	32.54
Hartmann (1998)	0.22	11.70	–	35.39
Nonobe, Ibaraki (1999)	–	–	–	35.86
Bouleimen, Lecocq (2003)	0.23	11.90	–	37.68
Kolish (1996)	1.29	13.23	–	38.75

Table 3
Results of the LNS search with an ILP solver for the subproblem resolution.

Prob.	av. (max) ΔUB	av. ΔLB	# best	av. CPU
KSD30	0.08(3.17)	–	457/480	165.04 (1485)
KSD60	0.63(7.07)	11.47	371/480	397.39 (2238)

huge number of partial solutions explored by the solver during the search for the optimal subproblem solution. On the KSD instance j1201_1.sm, the number of fails encountered during the search varies from 7 to 20000! This explains the rather large CPU times of our method.

It can be noted that Valls, Ballestin, and Quintanilla (2002) obtain on the KSD 120 set better results (32.04% above CPM) within a smaller amount of time (4.01 seconds on average) when they generate 10000 schedules. However we underline the simplicity, the ease of implementation and the generality of our method.

To underline this generality, we replace the CP solver by the ILP CPLEX solver for the subproblem resolution. The preprocessing made on the ILP is the same as for the CP solver (see section 3.4). We set the branch and bound process to Depth First Search and we select the parameter "emphasis on feasibility." We set the maximum time allotted to the solver to $H := 5$ s. The results we obtain on the KSD 30 and 60 sets are displayed in table 3. The table shows that even if the latter parameter increases dramatically the CPU times, the average deviation from the best solution is nearly the same as for our best results displayed in table 2. This enlightens the practical interest of the approach and seems to indicate that the choice of the subproblem generation method is more crucial than the method used to solve it, w.r.t. the quality of the obtained solutions.

Next, we compare the different selection strategies for the subproblem construction. Table 4 reports the average deviation above the optimal makespan on the KSD 30 set obtained by each strategy using ILOG SCHEDULER as subproblem resolution method. The results clearly show that the block strategy outperforms the other ones.

Since our method has several interacting components, we have to compare our results with the ones obtained by each component used separately as a resolution method. Namely, we compare our method with the 2 following heuristics:

Table 4
Comparison of the selection strategies on the KSD 30 set.

Method	Δ opt.
Random&Critical	1.5338
Random&Project Predecessors	1.2370
Random&Contiguous Activities	1.0885
Random&All Predecessors	0.9524
Block	0.02

Table 5
Comparison with SCHEDALONE and FBALONE.

Method	av. (max) ΔUB		av. ΔLB	av. (max) CPU	
	KSD30	KSD60	(KSD60)	KSD30	KSD60
SCHEDALONE	0.17 (11.95)	1.68 (20.13)	13.03	12.77 (236)	65.73 (427)
FBALONE	0.32 (8.62)	1.10 (16.03)	12.12	123 (123)	223 (223)
LSSPER	0 (0)	0.22 (3.54)	10.81	10.26 (123)	38.78 (223)

- SCHEDALONE: The CP solver (ILOG SCHEDULER) is used to solve the global problem as a truncated implicit enumeration method (using the same branching scheme as for the subproblem resolution).
- FBALONE: The subproblem resolution phase is removed. The method amounts to a multi-start heuristic where the forward/backward procedure is applied at each iteration.

For a fair comparison, we stop SCHEDALONE and FBALONE when the maximal time spent by the LNS method on the same set of instances is reached (for SCHEDALONE the maximal time spent sometimes exceeds this value due to the time necessary to find next stopping solution). Table 5 gives the results of LSSPER, SCHEDALONE and FBALONE on the KSD 30 and 60 instance sets.

The results show that the proposed LNS method performs better than both SCHEDALONE and FBALONE in terms of CPU times and quality of the solutions found. This underlines that the good results obtained by LSSPER are due to the cooperation between its components: exact resolution, local search and forward/backward heuristics.

We have also evaluated the impact of the characteristics of the instances on the performance of our method on the KSD 60 instance set. In the KSD sets, instances are characterised by the network complexity (NC), the resource factor (RF) (average number of resources required by the activities) and the resource strength (RS) (increasing in function of the average number of available resource units). As already shown in many other approaches, the hardest instances of these sets are the highly disjunctive ones (with small RS). The impact of this parameter is shown in figure 6. We can clearly note that the apparent difficulty of problems grows exponentially as RS decreases, while other parameters (NC and RF) do not seem to have a similar impact, as shown in figures 7

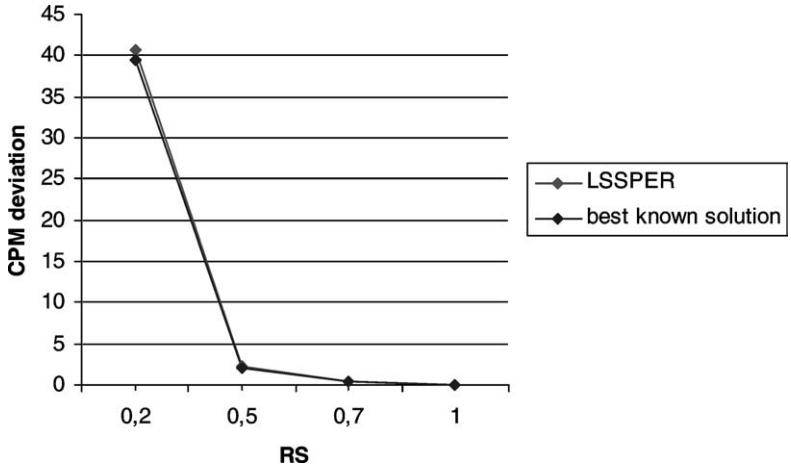

Figure 6. Deviation from CPM considering RS parameter (KSD 60).

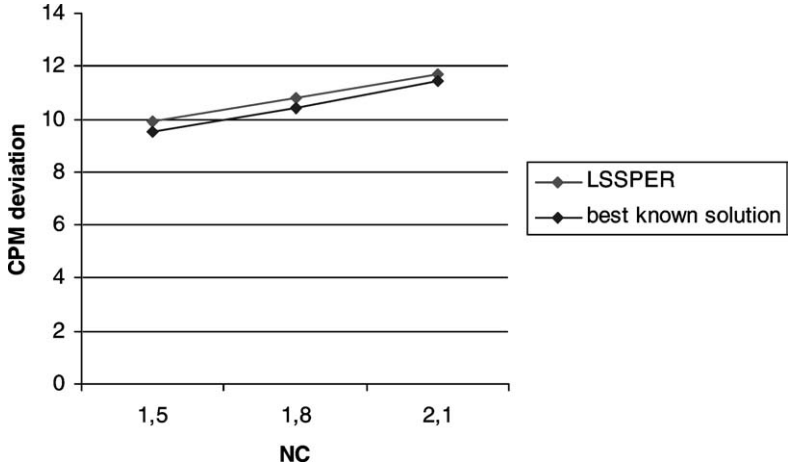

Figure 7. Deviation from CPM considering NC parameter (KSD 60).

and 8. However, we can remark the relative bad behaviour of the method, compared to best known solutions, on instances with RF $= 0.25$. This may indicate that the subproblem selection method is less suitable in this particular case where only one resource is required by an activity.

5. Conclusion

This paper presents LSSPER, a new competitive heuristic for solving the RCPSP, exploiting local search and exact resolution principles. At each iteration, the method considers a varying size large neighbourhood, made up of the set of solutions to a given subproblem, that is explored with the help of a truncated exact method. The solution

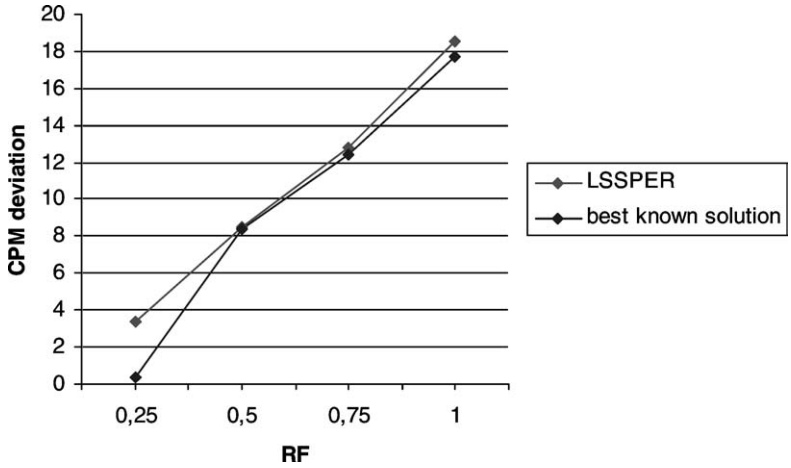

Figure 8. Deviation from CPM considering RF parameter (KSD 60).

provided by this partial optimisation is then re-injected into the current solution which is also post-optimised. The subproblem non-uniform generation procedure, combined with periodic re-start of the search, enables to introduce a diversification aspect in the exploration of the search space and to hopefully avoid local optimum.

LSSPER takes place in a more general context where each constituting module can be specified independently. Thus, this general method proposes an adaptable global framework for solving various combinatorial optimisation problems.

Notes

1. The selected activity i has the smallest latest finish time LF_i. LF_j's are computed by backward recursion in the project network (ignoring resources) after assigning LF_{n+1} to a given upper bound.
2. Results available at `http://www.bwl.uni-kiel.de/Prod/psplib/library.html`

References

Adams, J., Balas, E., and Zawack, D. (1988). "The Shifting Bottleneck Procedure for Job Shop Scheduling." *Management Science* 34, 391–401.

Ahuja, R.K., Ergun, O., Orlin, J.B., and Punnen, A.P. (2002). "A Survey of Very Large-Scale Neighborhood Search Techniques." *Discrete Applied Mathematics* 123(1–3), 75–102.

Alvarez-Valdés, R. and Tamarit, J.M. (1989). "Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis." In R. Słowiński and J. Weglarz (eds.), *Advances in Projet Scheduling*. Amsterdam: Elsevier, pp. 113–134.

Applegate, D. and Cook, B. (1991). "A Computational Study of the Job Shop Scheduling Problem." *ORSA Journal of Computing* 3(2), 149–156.

Artigues, C., Michelon, P., and Reusser, S. (2003). "Insertion Techniques for Static and Dynamic Resource-Constrained Project Scheduling." *European Journal of Operational Research* 149(2), 249–267.

Baar, T., Brucker, P., and Knust, S. (1998). "Tabu-Search Algorithms and Lower Bounds for the Resource-Constrained Project Scheduling Problem." In S. Voss, S. Martello, I. Osman, and C. Roucairol (eds.), *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, pp. 1–18.

Balas, E., Lancia, G., Serafini, P., and Vazacopoulos, A. (1998). "Job Shop Scheduling with Deadlines." *Journal of Combinatorial Optimization* 1(4), 329–353.

Baptiste, P. and Le Pape, C. (2000). "Constraint Propagation and Decomposition Techniques for Highly Disjunctive and Highly Cumulative Project Scheduling Problems." *Constraints* 5, 119–139.

Baptiste, P., Le Pape, C., and Nuijten, W. (1995). "Constraint-Based Optimization and Approximation for Job-Shop Scheduling." In *AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems*, Montreal.

Bell, C.E. and Han, J. (1991). "A New Heuristic Solution Method in Resource-Constrained Project Scheduling." *Naval Research Logistics* 38, 315–331.

Bent, R. and Van Hentenryck, P. (2001). "A Two-Stage Hybrid Local Search for the Vehicle Routing Problem with Time Windows." Technical Report CS-01-06, Brown University.

Bouleimen, K. and Lecocq, H. (2003). "A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem." *European Journal of Operational Research* 149(2), 249–267.

Brucker, P., Drexl, A., Möring, R., Neumann, K., and Pesch, E. (1999). "Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods." *European Journal of Operational Research* 112, 3–41.

Caseau, Y. and Laburthe, F. (1999). "Effective Forget-and-Extend Heuristics for Scheduling Problems." In *CP-AI-OR'99, Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Ferrara, Italy.

Cho, J.-H. and Kim, Y.-D. (1997). "A Simulated Annealing Algorithm for Resource Constrained Project Scheduling Problems." *Journal of the Operational Research Society* 48, 735–744.

Congram, R., Potts, C., and Van de Velde, S. (2002). "An Iterated Dynasearch Algorithm for the Single-Machine Total Weighted Tardiness Scheduling Problem." *INFORMS Journal on Computing* 14(1), 52–67.

Frangioni, A., Necciari, E., and Scutellà, M.G. (2004). "A Multi-Exchange Neighborhood for Minimum Makespan Machine Scheduling Problems." *Journal of Combinatorial Optimization* 8(2), 195–220.

Gendreau, M., Pesant, G., and Rousseau L.-M. (2002). "Using Constraint-Based Operators with Variable Neighborhood Search to Solve the Vehicle Routing Problem with Time Windows." *Journal of Heuristics* 8(1), 43–58.

Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic.

Hartmann, S. (1998). "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling." *Naval Research Logistics* 45, 733–750.

Hartmann, S. (1998). "A Self-Adapting Genetic Algorithm for Project Scheduling Under Resource Constraints." *Naval Research Logistics* 49, 433–448.

Hartmann, S. and Kolisch, R. (2000). "Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem." *European Journal of Operational Research* 127(2), 297–316.

Klein, R. and Scholl, A. (1999). "Computing Lower Bound by Destructive Improvement: An Application to Resource-Constrained Project Scheduling." *European Journal of Operational Research* 112, 322–346.

Kohlmorgen, U., Schmeck, H., and Haase, F. (1999). "Experiences with Fine-Grained Parallel Genetic Algorithms." *Annals of Operations Research* 90, 203–219.

Kolisch, R. (1996). "Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation." *European Journal of Operational Research* 90, 320–333.

Kolisch, R., Sprecher, A., and Drexl, A. (1998). "Benchmark Instances for Project Scheduling Problems." In J. Weglarz (ed.), *Handbook on Recent Advances in Project Scheduling*. Kluwer, pp. 197–212.

Kolisch, R. and Hartmann, S. (1999). "Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis." In J. Weglarz (ed.), *Project Scheduling: Recent Models, Algorithms and Applications*. Kluwer, pp. 147–178.

Kolisch, R. and Padman, R. (2001). "An Integrated Survey of Deterministic Project Scheduling." *Omega* 29(3), 249–272.

Lee, J.-K. and Kim, Y.-D. (1996). "Search Heuristics for Resource-Constrained Project Scheduling." *Journal of the Operational Research Society* 47, 678–689.

Leon, V. and Ramamoorthy, B. (1995). "Strength and Adaptability of Problem-Space Based Neighborhoods for Resource-Constrained Scheduling." *OR Spektrum* 17, 173–182.

Li, K.Y. and Willis, R.J. (1992). "An Iterative Scheduling Technique for Resource-Constrained Project Scheduling." *European Journal of Operational Research* 56, 370–379.

Mausser, H.E. and Lawrence, S.R. (1997). "Exploiting Block Structure to Improve Resource-Constrained Project Schedules." In F. Glover, I. Osman, and J. Kelley (eds.), *Metaheuristics 1995: State of the Art*. Maryland: Kluwer.

Mautor, T. and Michelon, P. (1997). "Mimausa: A New Hybrid Method Combining Exact Solution and Local Search." In *MIC'97, 2nd Metaheuristics International Conference*, Sophia Antipolis, France.

Mautor, T. and Michelon, P. (2001). "Mimausa: An Application of Referent Domain Optimization." Technical Report, Laboratoire d'Informatique d'Avignon.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). "Ant Colony Optimization for Resource-Constrained Project Scheduling." *IEEE Transactions on Evolutionary Computation* 6(4), 333–346.

Naphade, K.S., Wu, S., and Storer, S.D. (1997). "Problem Space Search Algorithms for Resource-Constrained Project Scheduling." *Annals of Operations Research* 70, 307–326.

Nonobe, K. and Ibaraki, T. (1999). "Formulation and Tabu Search Algorithm for the Resource-Constrained Projet Scheduling Problem (RCPSP)." Technical Report, Department of Applied Mathematis and Physics, Kyoto University, Japan.

Patterson, J. (1984). "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem." *Management Science* 30(7), 854–867.

Pinson, E., Prins, C., and Rullier, F. (1994). "Using Tabu Search for Solving the Resource-Constrained Project Scheduling Problem." In *International Workshop on Project Management and Scheduling, PMS'94*, Louvain, Belgique, pp. 102–106.

Pritsker, A.A.B., Waters, L.J., and Wolfe, P.M. (1969). "Multiproject Scheduling with Limited Resources: A Zero–One Programming Approach." *Management Science* 16, 93–107.

Sampson, S.E. and Weiss, E.N. (1993). "Local Search Techniques for the Generalized RCPSP." *Naval Research Logistic Quarterly* 40, 665–675.

Schirmer, A. (2000). "Case-Based Reasoning and Improved Adaptive Search for Project Scheduling." *Naval Research Logistics* 47, 201–222.

Schutten, J.M.J. (1998). "Practical Job Shop Scheduling." *Annals of Operations Research* 83, 161–177.

Shaw, P. (1998). "Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems." In *Principles and Practice of Constraint Programming CP-98*, Pisa, Italy, Lecture Notes in Computer Science, Vol. 1520, pp. 417–431.

Sourd, F. (2001). "Scheduling Tasks on Unrelated Machines: Large Neighborhood Improvement Procedures." *Journal of Heuristics* 7(6), 519–531.

Taillard, E. and Voss, S. (2002). "Popmusic – Partial Optimization Metaheuristic Under Special Intensification Conditions." In C.C. Ribeiro and P. Hansen (eds.), *Essays and Surveys in Metaheuristics*. Boston: Kluwer, pp. 613–629.

Thomas, P.R. and Salhi, S. (1998). "A Tabu Search Approach for the Resource Constrained Project Scheduling Problem." *Journal of Heuristics* 4, 123–139.

Valls, V., Ballestin, F., and Quintanilla, M.S. (2000). "Resource-Constraint Project Scheduling: A Critical Activity Reordering Heuristic." In *PMS'2000, 7th International Workshop on Project Management and Scheduling*, Osnabruck, Germany, pp. 282–283.

Valls, V., Ballestin, F., and Quintanilla, M.S. (2002). "A Hybrid Genetic Algorithm for the RCPSP with the Peak Crossover Operator." In *Proc. of 8th International Workshop on Project Management and Scheduling, PMS 2002*, Valencia, Spain, pp. 368–370.