Available online at www.sciencedirect.com
SCIENCE DIRECT®
European Journal of Operational Research 155 (2004) 402–413
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/dsw

Discrete Optimization

# Solving the resource-constrained project scheduling problem by a variable neighbourhood search

Krzysztof Fleszar [a], Khalil S. Hindi [b,*]

[a] *Institute of Control and Computation Engineering, Warsaw University of Technology, ul. Nowowiejska 15/19, 00-665 Warszawa, Poland*

[b] *Faculty of Engineering and Architecture and the School of Business, American University of Beirut (AUB), P.O. BOX 11-0236, Riad El Solh, Beirut 1107 2020, Lebanon*

Received 18 February 2000; accepted 19 September 2002

**Abstract**

The well-known, challenging problem of resource-constrained project scheduling is addressed. A solution scheme based on variable neighbourhood search is presented. The solution is coded by using activity sequences that are valid in terms of precedence constraints. The sequences are turned into valid active schedules through a serial scheduler. The search of the solution space is carried out via generating valid sequences using two types of move strategy.

Much of the power of the solution scheme is attributable to repeatedly employing effective lower bounding and precedence augmentation, both of which serve to reduce the solution space.

The effectiveness of the solution scheme is demonstrated through extensive experimentation with a standard set of 2040 benchmark problem instances. The best-known solutions have been improved upon for 48 instances and the best-known lower bounds have also been improved upon for 148 problem instances. The results are inferior to the best-known for a relatively small number of problem instances, but even then the average deviation is very small indeed.


*Keywords:* Scheduling; Variable neighbourhood search; Heuristics

## 1. Introduction

The resource-constrained project scheduling problem (RCPSP) [9] is concerned with $n$ non-preemptive activities $j = 1, 2, \ldots, n$ and $m$ renewable resources. The availability of each resource is $R_k, k = 1, 2, \ldots, m$. An activity $j$ has to be processed for $d_j$ time units. During this time a constant amount $r_{jk}$ of resource $k$ is occupied. Each activity $j$ has a set of immediate predecessors $P_j$ and a set of immediate successors $S_j$. It is enough to define either sets of immediate successors or sets of immediate predecessors only, because $i \in P_j \iff j \in S_i$. If $i \in P_j$, then we say $i \prec j$ ($i$ precedes $j$) and an activity $j$ cannot be started before $i$ is finished. Moreover, if $i \prec j$ and $j \prec l$, then also $i \prec l$. It is often assumed that the first activity and the last activity are dummy activities, i.e., they have zero

* Corresponding author. Tel.: +961-1-350000x3486.
*E-mail addresses:* k.fleszar@elka.pw.edu.pl (K. Fleszar), khalil.hindi@aub.edu.lb (K.S. Hindi).
*URLs:* http://www.ia.pw.edu.pl/~kfleszar, http://www.brunel.ac.uk/~emstksh.

0377-2217/$ - see front matter © 2003 Published by Elsevier B.V.
doi:10.1016/S0377-2217(02)00884-6

duration and zero resource consumption, and correspond respectively to the 'project start' and the 'project end'. The objective is to find for each activity $j$ its start time $s_j$ ($s_1 = 0$) such that the makespan of the project $T = s_n$ is minimised.

RCPSP can be represented formally as the following binary integer program [19,21]:

$$\min \sum_{t=\mathrm{EF}_n}^{\mathscr{T}} t x_{nt} \tag{1}$$

subject to

$$\sum_{t=\mathrm{EF}_j}^{\mathscr{T}} x_{jt} = 1 \quad \forall j, \tag{2}$$

$$\sum_{t=\mathrm{EF}_j}^{\mathscr{T}} x_{jt}(t - d_j) \geqslant \sum_{t=\mathrm{EF}_i}^{\mathscr{T}} t x_{it} \quad \forall j, i \in P_j, \tag{3}$$

$$\sum_{j=1}^{n} \left( r_{jk} \sum_{q=t}^{t+d_j-1} x_{jq} \right) \leqslant R_k \quad \forall k, t = 1, \ldots, \mathscr{T}, \tag{4}$$

$$x_{jt} \in \{0, 1\} \quad \forall j, t; \tag{5}$$

where the decision variable

$$x_{jt} = \begin{cases} 1 & \text{if activity } j \text{ is completed in period } t, \\ 0 & \text{otherwise,} \end{cases}$$

and the additional parameters are as follows: $\mathrm{EF}_j$ is the earliest finish time of activity $j$, computed by a forward critical path pass on the project network; $\mathscr{T}$ is the upper bound on horizon.

Constraints (2) ensure that activity $j$ can finish in one and only one time period, while constraints (3) ensure that precedence relationships between any two activities are not violated and constraints (4) ensure that no resource is used beyond its capacity.

A considerable amount of research on RCPSP has appeared. The problem is well recognised as important and difficult. It is easy to see that it is a generalisation of the classical job shop scheduling problem, and is thus NP-hard [2]. Even for moderately sized problems, finding an optimal solution in a reasonable amount of time can be very difficult. This has motivated research on various solution methods. Excellent reviews are available [1,4,20]. Invaluable recent sources are [3,10,23].

In this paper, we present an effective heuristic solution scheme for the problem, based on a variable neighbourhood search (VNS) enhanced by effective lower bounding and precedence augmentation techniques.

## 2. VNS for RCPSP

### 2.1. Schedule representation

A schedule is represented, as in [13,17], by a valid sequence of activities $a = (a_1, a_2, \ldots, a_n)$. A valid sequence is one that satisfies the precedence constraints:

$$\forall i = 1, 2, \ldots, n, \quad P_{a_i} \subseteq \{a_1, a_2, \ldots, a_{i-1}\} \wedge S_{a_i} \subseteq \{a_{i+1}, a_{i+2}, \ldots, a_n\} \tag{6}$$

To obtain start times $s_j$ from the sequence $a$, the serial schedule generation scheme [13], shown in Fig. 1, is applied. This scheme plays an important role in classical machine scheduling, where it is known as list scheduling [11,12]. It always gives an active schedule; i.e., a schedule in which it is not possible to left shift any activity without violating the constraints [22]. It also ensures that for each active schedule there exists an appropriate corresponding sequence. However, a disadvantage of this representation is that there is no one to one correspondence between sequences and schedules, in that the same schedule may be produced from more than one unique sequence, with the result that the space of representation is greater than the space of active schedules.

### 2.2. Neighbourhood search

Neighbourhood search (NS) is an algorithm aimed at finding a local minimum starting from a given point in the solution space. The neighbourhood of the current point is searched for better points and if one is found, it is adopted as the new starting point. The process is repeated until a local optimum is found; i.e., until no improving move can be found.

```
procedure Serial sequence scheduler(a) begin
    Initialise resource availability profile;
    for i = 1 to n do begin
        j = a_i;
        t = max{0, max{s_l + d_l | l ∈ P_j}};
        Find t' ≥ t, the earliest time at which activity j can be
            scheduled without exceeding remaining resources
            throughout its duration;
        Schedule j to start at s_j = t' and update the resource
            availability profile accordingly;
    end
end;
```

Fig. 1. Serial sequence scheduler procedure.

*Neighbourhood and moves*: In the present context, a point in the solution space is represented by the corresponding valid sequence and the *neighbourhood* of a valid sequence is defined as the set of valid sequences that result from moving one activity to a new position.

If only direct predecessors and successors are taken into consideration, then a move (shift) in the neighbourhood can be defined such that an activity $a_i$ may be moved to a new position $i'$:

$$\mathrm{LL}(a_i) \leqslant i' \leqslant \mathrm{RL}(a_i) \tag{7}$$

where the *left limit* $\mathrm{LL}(a_i)$ and *right limit* $\mathrm{RL}(a_i)$ are defined by

$$\begin{aligned} \mathrm{LL}(j) &= \max\{l | a_l \in P_j\} + 1, \\ \mathrm{RL}(j) &= \min\{l | a_l \in S_j\} - 1. \end{aligned} \tag{8}$$

In this move strategy [13], if $a_i$ is moved to $i' < i$, then the activities $a_{i'}, a_{i'+1}, \ldots, a_{i-1}$ are shifted rightwards by one position. Similarly, if $a_i$ is moved to $i' > i$, then the activities $a_{i+1}, a_{i+2}, \ldots, a_{i'}$ are shifted leftwards by one position.

However, we have developed a more forceful move strategy by taking account of both direct and indirect predecessors and successors. Let $\overline{P}_j$ be the set of all direct and indirect predecessors of $j$ and $\overline{S}_j$ be the set of all direct and indirect successors of $j$. A move in this strategy, which we term an 'enhanced move', can then be defined such that an activity $a_i$ may be placed in a new position $i'$:

$$\mathrm{ELL}(a_i) \leqslant i' \leqslant \mathrm{ERL}(a_i) \tag{9}$$

where $\mathrm{ELL}(a_i)$ and $\mathrm{ERL}(a_i)$ are the *extended left limit* and the *extended right limit* respectively, defined by

$$\begin{aligned} \mathrm{ELL}(j) &= \left|\overline{P}_j\right| + 1, \\ \mathrm{ERL}(j) &= \left|\overline{S}_j\right| - 1. \end{aligned} \tag{10}$$

In order to preserve precedence relations, an enhanced move of activity $j = a_i$ to the left (right) forces all its predecessors $\overline{P}_j$ (successors $\overline{S}_j$) to move to the left (right), while activities that are not in any precedence relation with $j$ 'jump' to the other side of $j$. In the example of Fig. 2, the predecessors and successors, both direct and indirect, of activity 4 are marked with $'$. The first move of activity 4 to the left simply leads to activities 4 and 5 exchanging positions. The second move forces activities 2 and 3 to move to the left, because they have to precede 4. The nearest activity at the left side that is not in a precedence relation to 4 is 6 and it 'jumps' to the other side of 4. Note, that

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Starting sequence | $1'$ | 6 | $3'$ | $2'$ | 5 | **4** | $8'$ | 7 | $9'$ |
| First move to the left | $1'$ | 6 | $3'$ | $2'$ | **4** | 5 | $8'$ | 7 | $9'$ |
| Second move to the left | $1'$ | $3'$ | $2'$ | **4** | 6 | 5 | $8'$ | 7 | $9'$ |
| First move to the right | $1'$ | $3'$ | 6 | $2'$ | 5 | 7 | **4** | $8'$ | $9'$ |

Fig. 2. An example of enhanced moves.

activities 2 and 3 remain in the same order. The only possible move to the right forces activity 8 to move to the right and activity 7 'jumps' to the other side of 4.

It is clear that the neighbourhood defined by the enhanced move strategy is larger than that defined by the simple move strategy, in the sense that several simple moves may be needed to reach a point moved to by a single enhanced move. Thus the enhanced move strategy helps diversify the search and has proved in our experimentation to lead to distinctly better results overall; albeit at the expense of increasing processing time.

It is worth noting, however, that, in either move strategy, not every move leads to a new schedule. For example, if activity $a_i$ is moved to position $i+1$ and activities $a_i$ and $a_{i+1}$ do not compete for resources, then the new schedule will still be the same. To find out whether a move does lead to a change in schedule, it is sufficient to recalculate only the new start times of the activities that have changed positions. If none changes, then the overall schedule will remain the same and there is no point in computing it. This leads to significant computation time saving when looking for the best move of an activity.

*Restricted NS*: The worth of a move is calculated as the decrease in project makespan resulting from it, with moves giving rise to larger decreases being preferable.

The restricted NS scheme could be described as follows. The sequence of $n$ activities is searched as if it were a circular array. Let $\alpha$ be a number less than $n$, such that dividing $n$ by $\alpha$ leads to a suitable number of subdivisions with a remainder. Then each activity is taken in turn and all the legal moves involving it evaluated and so on until $\alpha$ activities are processed. If the best move discovered does lead to a decrease in project makespan, then it is effected. Thereafter, the search is resumed from where it reached and so on, processing the sequence of activities in a circular fashion. The process terminates, when a total of $n$ activities are processed without discovering a makespan-reducing move, which proves that the current solution is a local optimum.

An effective choice of $\alpha$ has been found by experimentation to be $\alpha = \min\{7, n\}$.

Three points regarding this scheme are noteworthy:

1. The procedure is akin to a descent search with a candidate list strategy, in the sense that the legal moves involving the next $\alpha$ activities in the sequence are those on the candidate list at each stage. In this way, a great deal of computation time is saved in comparison with steepest descent search.
2. A measure of search diversification is introduced by the fact that as the search progresses, the composition of the candidate lists changes, as a result of searching the sequence of activities in a circular fashion, with $n/\alpha$ having a remainder.
3. The procedure is also akin to Tabu search, in the sense that when a move is effected, the activity concerned is unlikely to be moved back to its old position until it is encountered again while searching the sequence (variable Tabu duration), or unless the resulting sequence leads to the best makespan so far (aspiration by objective). Thus this search scheme has all the advantages of basic Tabu search.

### 2.3. Initial solution

Extensive experimentation has shown that the better the initial solution with which VNS for RCPSP is started, the better the final solution will be on average. Moreover, on average, much shorter computation times are needed.

The starting sequence is generated using well-known, one-pass, priority-rule-based heuristics [13], which build a schedule by adding activities one by one. At each step, starting from the partial schedule assembled thus far, a decision set $D$, containing the activities that have all their predecessors already scheduled and thus can be added to the schedule, is calculated. For each activity $j \in D$, a priority $v_j$ is calculated and the activity with the highest (or lowest) priority is selected. Six priority rules for the serial schedule generation scheme may be used [13]: greatest rank positional weight (GRPW), latest finish time (LFT), latest start time (LST), minimum slack (MSLK), most total successors (MTS) and shortest processing time (SPT). Table 1 summarises the rules we have used. $\mathrm{LF}_j$

Table 1
Priority rules

| Rule | Ref. | $v_j$ |
|---|---|---|
| GRPW | [1] | $d_j + \sum_{i \in S_j} d_i$ |
| LFT | [4] | $\mathrm{LF}_j$ |
| LST | [16] | $\mathrm{LF}_j - d_j$ |
| MSLK | [4] | $\mathrm{LF}_j - \mathrm{EF}_j$ |
| MTS | [1] | $\lvert S_j \rvert$ |
| SPT | [1] | $d_j$ |

denotes the latest finishing time and is calculated by backward recursion on the basis of the current upper bound [5], which is initially achieved by applying one of the priority rules that do not require $\mathrm{LF}_j$; e.g., GRPW. $\mathrm{EF}_j$ is the earliest finishing time calculated on the basis of both precedence and resource constraints and updated in accordance with the current partial schedule.

Three strategies for generating an initial solution have been experimented with:

1. Use each priority rule to generate one sequence and adopt the best.
2. Use each priority rule to generate a sequence $a$, apply the restricted NS algorithm, $\mathrm{NS}(a)$, to find a local optimum. Adopt the best sequence overall.
3. Apply sampling based on priority rules [13], followed by $\mathrm{NS}(a)$.

The third strategy, which proved to be the most effective, resorts to generating sequences from the priority rules cyclicly. Each time a new sequence is generated, NS is applied and the best sequence overall is adopted. Processing is stopped when a number of cycles elapses without generating a better sequence. In our implementation, we fixed this number to 2, on the basis of some experimentation. Obviously, the procedure is meaningful only if different sequences can be generated from the application of the same priority rule. This is done by choosing the next activity from $D$ with a probability related to its priority value, rather than according to the priority value itself. Let $p_j$ be the probability of choosing activity $j$ which has a priority value $v_j$. Then $p_j$ is calculated as follows for rules that select the activity with the highest priority value:

$$r_j = v_j - \min_{i \in D} v_i, \qquad p_j = \frac{r_j^2}{\sum_{i \in D} r_j^2}. \tag{11}$$

For rules that select the activity with the lowest priority value, $v_j$ is replaced with $-v_j$. It is worth noting that this sampling strategy is one of regret-based biased random sampling [7,16]. However, it differs from the usual in squaring the regret values, $r_j$. This is done in order to give a strong preference to the activities with higher priority values, which in turn makes the sampling follow the original heuristics more closely.

### 2.4. Variable neighbourhood search

VNS is a metaheuristic developed by Hansen and Mladenović [6,18], designed to find near-optimal solutions. Let $N_k$, $k = 1, 2, \ldots, k_{\max}$, be a set of predefined neighbourhood structures. The search is started with an initial solution, corresponding to a local optimum, from $k = 1$. The following steps are carried out repeatedly until a stopping criterion is met:

*Shaking*: A random point from the $N_k$ neighbourhood of the current point is generated.
*Local search*: A local optimum is found starting from the new point generated in the previous step.
*Move or not*: If the new local optimum is better than the previous one, it becomes the current solution and the processing is restarted. Otherwise $k$ is increased by one.

For the problem in hand, we define the $N_k$ neighbourhood of a sequence $a$ as the set of sequences that can be generated starting from $a$ by making $k$ legal moves of $k$ distinct activities. Thus, randomly generating a sequence from neighbourhood $N_k$ is carried out by a series of $k$ random legal moves of $k$ randomly selected activities.

The VNS developed in this work is shown in Fig. 3, where $\mathrm{NS}(a)$ is the procedure implementing the restricted NS for finding a local optimum, as described above; $\mathrm{MK}(a)$ is the makespan

```
procedure VNS(a) begin
    k = 1;
    Generate first sequence a
    NS(a);
    while k ≤ k_max do begin
        repeat k times begin
            Select random a' from N_k of a;
            NS(a');
            if a' is better than a then begin
                a = a'; k = 1;
                if MK(a) = LB then stop;
                Restart while loop;
            end
        end;
        k = k + 1;
    end
end;
```

Fig. 3. VNS procedure.

corresponding to sequence $a$ and LB is the best (highest) lower bound discovered so far.

This algorithm differs from that suggested by Hansen and Mladenović by repeating the search $k$ times for each value of $k$, thus increasing the chance of finding a good local optimum when the neighbourhood is large.

The stopping criterion employed here is to terminate the search whenever a solution with a makespan equal to a known lower bound is achieved or when $k > k_{\max}$.

## 3. Lower bounds and precedence augmentation

Let the *head*, $h_j$, of activity $j$ be the time that for some reason has to pass before $j$ can be started. Analogously, let the *tail*, $t_j$, be the time that for some reason has to pass after finishing activity $j$ before the project is completed.

Two activities $i$ and $j$ are *incompatible*, if they cannot be processed in parallel. This may be due to a precedence relation (direct or indirect) or due to excessive resource consumption when $\exists k | r_{ik} + r_{jk} > R_k$.

Several lower bounding techniques, most developed by Robert Klein and Armin Scholl [12], have been used: critical path bound (relaxation of resource constraints), capacity bound (relaxation of precedence relations), one-machine bound (relaxation of resource constraints to one resource-machine), two-machine bound (another version of relaxation of resource constraints, this time to two parallel machines) and destructive improvement. This latter approach [12] is based on computing sharp time windows (maximising heads and tails), using so called reduction techniques. From amongst the latter, we have employed reduction by forward and backward pass (i.e., CPM), reduction by subprojects and reduction by core times. The sharper time windows are then used in contradiction arguments in an attempt to improve the value of the lower bound.

Having calculated good heads and tails and obtained a project schedule, with a makespan equal to $T$, we can embark on a process of *precedence augmentation*, by which we denote the procedure that permanently introduces new precedence relations into the project. Since we are interested only in schedules with a makespan that is shorter than $T$, we can assume an upper bound $\mathrm{UB} = T - 1$. To add new precedences, all pairs of activities that are not in any precedence relation are considered. The procedure for such a pair is as follows:

1. If $h_j + t_i \geqslant \mathrm{UB}$, i.e., if $i$ has to be finished before $j$ can start, then add $i \prec j$.
2. If $i$ and $j$ are incompatible and $h_j + d_j + t_i + d_i > \mathrm{UB}$, i.e., if enforcing $j \prec i$ leads to a makespan larger than UB, then add $i \prec j$.
3. As a trial, implicitly impose $i \not\prec j$ and calculate the resulting heads and tails:
   3.1. if $i$ and $j$ are incompatible (in which case $i \not\prec j \Rightarrow j \prec i$), then
   $h_i = \max\{h_i, h_j + d_j\}$,
   $t_j = \max\{t_j, t_i + d_i\}$;
   3.2. if $i$ and $j$ are compatible (in which case $i \not\prec j \Rightarrow j \prec i \vee i$ and $j$ overlap), then
   $h_i = \max\{h_i, h_j - d_i + 1\}$,
   $t_j = \max\{t_j, t_i - d_j + 1\}$.

If any head or tail is thereby improved, then with the improved values, calculate the resulting lower bound LB′. If $\mathrm{LB}' > \mathrm{UB}$, then add $i \prec j$.

Since there are $O(n^2)$ pairs of activities to be considered, computation of the lower bound in

step 3 should be fast, yet effective. Extensive experimentation showed that both purposes are well served by using time lags saved from the previous reduction by subprojects procedure to update heads and tails, followed by reduction by core times. During the latter procedure, contradiction may occur ($LB' > UB$), in which case a new precedence is added.

Precedence augmentation is clearly advantageous since it reduces the solution space, particularly that adding one precedence relation results in additional transitive precedence relations. Reduction of the solution space is manifested in narrowing the limits of moves in NS, which, in turn, has the benefit of reducing the processing time of VNS. However, in some cases the sequence corresponding to the best solution found so far may violate the new precedences. If this happens, the sequence must be repaired (see the *repair algorithm* of Fig. 4) before resuming the NS. The local minimum found in this way is sometimes better than the incumbent.

```
procedure Repair(a) begin
    i := 1;
    while i < n do
        if a_i has predecessors to the right then begin
            Find a_k, the right most predecessor of a_i;
            s := a_i;
            for j := i + 1 to k do a_{j-1} := a_j;
            a_k := s;
        end
        else i := i + 1;
end;
```

Fig. 4. Repair sequence algorithm.

## 4. Overall scheme

The overall solution scheme is shown in Fig. 5. The solution process is stopped whenever the incumbent solution is proven to be optimal (makespan = lower bound); otherwise it is repeated until two runs in a row fail to yield improvement. A distinctive feature is the repeated attempts to narrow the gap between the upper bound and the lower bound by improving both. Moreover, another attempt is made at the end to improve the

```
procedure Augment begin
    while current solution is better than incumbent do begin
        Incumbent := current solution;
        if incumbent is optimal then stop;
        repeat
            Do precedence augmentation and recalculate lower bound;
            if lower bound is improved then
                if incumbent is optimal then stop;
        until no more precedences can be added;
        Repair current sequence and carry out neighbourhood search;
    end
end;

Calculate lower bounds;
Makespan of the incumbent := M (large number);
repeat
    Generate initial (current) solution (section 2.3);
    Augment;
    Current solution := VNS solution starting from current solution;
    Augment;
    if makespan has improved then β := 0 else β := β + 1;
until β = 2;
Attempt lower bound improvement (figure 6 );
```

Fig. 5. Overall scheme.

```
UB = makespan of the incumbent;
while LB + 1 < UB do begin
    Trial makespan := ⌊(LB + UB)/2 − 1⌋;
    Attempt to contradict the trial makespan with precedence
        augmentation and recalculation of lower bounds;
    if contradicted then LB := trial makespan + 1;
    else UB := trial makespan;
end;
```

Fig. 6. Lower bound improvement algorithm.

lower bound using the algorithm shown in Fig. 6, in order to narrow the gap further and improve confidence in the upper bound solution. In the VNS, $k_{\max}$ was set to 10.

## 5. Computational testing

The algorithm has been coded in Borland Delphi Pascal version 5 and all the tests have been performed on a Pentium III 1 GHz running under Windows 2000.

Although the algorithm has been tested on the Patterson data sets [20], we do not include the results here, due to the fact that the best of our solution schemes finds the optimal solutions for all instances and in only a few of them (13.3%) optimality is not verified by coincidence with the lower bound.

The four benchmark data sets of Kolisch et al. [14] proved more taxing. These sets, referred to as J30, J60, J90 and J120, consist of RCPSP instances with 30, 60, 90 and 120 activities respectively. The best-known results for these problem instances have been compiled from a variety of sources using many different algorithms [15], and can be downloaded from: ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/datasm.html.

Table 2 presents the results for the possible combinations of the various features of our algorithm; i.e., for the initial solution with and without enhanced moves, for VNS, and for VNS combined with augmentation, and with enhanced moves separately, as well as with both jointly. The solutions and the lower bounds are compared respectively with the best-known solutions and the best-known lower bounds. The 'VNS' and 'VNS + Aug.' algorithms are started from the initial solutions without enhanced moves, whereas 'VNS + Enh.' and 'VNS + Enh. + Aug.' are started from initial solutions with enhanced moves.

Initial solutions are quite good and VNS effects significant improvement. Enhanced moves improve solution quality considerably, but at the expense of increasing processing time. Precedence augmentation, on the other hand, does not seem to have an appreciable impact on solution quality, but improves lower bounds; thereby increasing chances of solution verification. Again, employing precedence augmentation increases processing time. However, for small problem instances belonging to the J30 class, introducing augmentation to VNS + Enh. decreases average processing time, since the time spent on augmentation is more than offset by the saving due to optimality verification.

The quality of solutions and lower bounds achieved by VNS with enhanced moves and precedence augmentation is quite impressive. This can be seen from the average and maximum relative deviations from the best known (Table 2) and from comparisons on the basis of the numbers of problem instances (Table 3). Almost all solutions for J30 are the same as the best-known ones and due to the fact that the best-known solutions are optimal, the overwhelming majority of our solutions is also optimal. For J60 and J90, the algorithm performs remarkably well in comparison with the best-known solutions. For J120, about 50% of the solutions are worse, but the average deviation is still less than 0.7%.

Table 2 indicates that employing precedence augmentation and enhanced moves increases processing times appreciably. This is all the more

Table 2
Deviations from best-known solutions and lower bounds, and processing times (+Enh. = with enhanced moves, +Aug. = with precedence augmentation)

| Set | Algorithm | Sol. dev. (%) | | LB dev. (%) | | Time (seconds) | |
|---|---|---|---|---|---|---|---|
| | | Av. | Max. | Av. | Max. | Av. | Max. |
| J30 | Initial Sol. | 0.48 | 6.90 | 1.81 | 15.87 | 0.07 | 0.17 |
| | Initial Sol. + Enh. | 0.24 | 4.41 | 1.81 | 17.87 | 0.09 | 0.47 |
| | VNS | 0.05 | 2.44 | 1.76 | 15.87 | 0.30 | 2.30 |
| | VNS + Aug. | 0.03 | 1.72 | 0.64 | 12.07 | 0.37 | 2.95 |
| | VNS + Enh. | 0.01 | 1.49 | 1.76 | 15.87 | 0.71 | 5.94 |
| | VNS + Enh. + Aug. | 0.01 | 1.56 | 0.64 | 12.07 | 0.64 | 5.86 |
| J60 | Initial Sol. | 1.61 | 12.50 | 0.85 | 10.48 | 0.32 | 1.00 |
| | Initial Sol. + Enh. | 1.01 | 8.05 | 0.85 | 10.48 | 0.59 | 3.38 |
| | VNS | 0.26 | 3.60 | 0.83 | 10.48 | 2.32 | 18.91 |
| | VNS + Aug. | 0.31 | 4.49 | 0.21 | 10.45 | 4.51 | 41.20 |
| | VNS + Enh. | 0.15 | 3.09 | 0.83 | 10.48 | 7.09 | 75.77 |
| | VNS + Enh. + Aug. | 0.14 | 3.28 | 0.15 | 10.45 | 8.89 | 80.70 |
| J90 | Initial Sol. | 1.91 | 11.70 | 0.49 | 7.92 | 0.98 | 3.14 |
| | Initial Sol. + Enh. | 1.36 | 10.00 | 0.49 | 7.92 | 1.86 | 9.83 |
| | VNS | 0.40 | 6.36 | 0.49 | 7.92 | 7.44 | 64.27 |
| | VNS + Aug. | 0.42 | 6.36 | 0.20 | 5.94 | 16.53 | 182.44 |
| | VNS + Enh. | 0.27 | 4.11 | 0.49 | 7.92 | 25.55 | 212.97 |
| | VNS + Enh. + Aug. | 0.23 | 3.85 | 0.20 | 5.94 | 32.43 | 247.91 |
| J120 | Initial Sol. | 5.55 | 16.00 | 0.75 | 8.30 | 3.44 | 9.02 |
| | Initial Sol. + Enh. | 3.62 | 9.82 | 0.75 | 8.30 | 9.41 | 37.42 |
| | VNS | 1.21 | 7.56 | 0.74 | 8.30 | 40.85 | 175.78 |
| | VNS + Aug. | 1.27 | 7.56 | 0.39 | 8.30 | 97.93 | 883.39 |
| | VNS + Enh. | 0.73 | 3.92 | 0.74 | 8.30 | 174.43 | 1088.78 |
| | VNS + Enh. + Aug. | 0.69 | 4.08 | 0.38 | 8.30 | 219.86 | 1126.97 |

Table 3
Results for VNS with enhanced moves and augmentation

| Set | Instances | Sol. vs best-known | | | LB vs best-known | | |
|---|---|---|---|---|---|---|---|
| | | Better | Same | Worse | Better | Same | Worse |
| J30 | 480 | 0 | 476 | 4 | 0 | 398 | 82 |
| J60 | 480 | 11 | 415 | 54 | 53 | 369 | 58 |
| J90 | 480 | 7 | 396 | 77 | 30 | 372 | 78 |
| J120 | 600 | 30 | 279 | 291 | 65 | 304 | 231 |

remarkable for the J120 set, which is known to be much more difficult than the other three sets, because the resource strength for it is lower on average (for more details see [14]). Nevertheless, processing times are reasonable, not exceeding 20 minutes over all the problem instances tackled.

Comparison of the lower bounds achieved by our algorithm with the best known lower bounds is also favourable. Since the lower bounds for the J30 set are optimal, it is not possible to find any better values. However, for all other sets, it was possible to find better lower bounds for a number of instances, but for all sets, some obtained lower bounds are inferior. However, even for the J120 set, where the number of instances with an inferior lower bound is largest, the average deviation is still less than 0.4%.

Table 4 shows for each set the number of solutions that have been verified to be optimal and the deviation between the makespan and the lower bound obtained by VNS with enhanced moves and augmentation. The deviation is quite low for J30 but grows rapidly with the size of the problem instance.

To ease comparison with other heuristic algorithms, Table 5 shows the average deviations of our solutions from the critical-path-based lower bound, and the number of schedules generated. It also shows the average deviation for the best-known solutions and the solutions obtained by the heuristic identified in [13] to be the most effective; i.e., GA—activity list [8]. It can be seen that for both sets of problem instances considered, the average deviation achieved by our best solution scheme is better than that achieved by GA—activity list and is very near that achieved by the best-known solutions.

However, as expected in a NS-based solution scheme, the number of complete schedules generated by our solution procedure is quite large. Since in our implementation, we build a large number of partial schedules, the number of complete schedules was calculated by dividing the number of tasks scheduled by the number of the tasks in the problem instance concerned. It is worth noting that it is customary to compare heuristics by restricting each to generating the same number of schedules; normally 1000 or 5000. But this is not possible in the present context, since even one local NS quite often involves generating more than 5000 schedules. In any case, it is a moot point whether raising the ceiling on the number of schedules that a particular heuristic is allowed to produce would help it discover better solutions, since that depends on the ability of the heuristic concerned to search the solution space effectively.

As can be seen from Table 3, we have been able to improve upon the best-known solutions and/or the best-known lower bounds for a number of instances. The instances for which the solutions

Table 4
Solutions vs lower bounds obtained by VNS with enhanced moves and augmentation

| Set | Instances | Verified solutions | Verified (%) | Dev. (%) | |
|---|---|---|---|---|---|
| | | | | Av. | Max |
| J30 | 480 | 397 | 82.7 | 0.69 | 14.04 |
| J60 | 480 | 364 | 75.8 | 2.50 | 22.50 |
| J90 | 480 | 367 | 76.5 | 2.81 | 21.05 |
| J120 | 600 | 214 | 35.7 | 7.11 | 27.74 |

Table 5
Solution deviation from critical-path lower bound and number of schedules generated

| Set | Algorithm | Dev. (%) | Schedules generated | |
|---|---|---|---|---|
| | | Av. | Av. | Max. |
| J60 | Initial Sol. | 12.98 | 1927 | 14,585 |
| | Initial Sol. + Enh. | 12.18 | 9272 | 79,823 |
| | VNS | 11.13 | 39,500 | 393,327 |
| | VNS + Aug. | 11.20 | 37,153 | 393,327 |
| | VNS + Enh. | 10.96 | 164,676 | 1,653,641 |
| | VNS + Enh. + Aug. | 10.94 | 152,503 | 1,653,641 |
| | Best-known solutions | 10.76 | – | – |
| | GA—activity list | 11.89 | – | 5000 |
| J120 | Initial Sol. | 40.02 | 11,947 | 69,763 |
| | Initial Sol. + Enh. | 37.38 | 88,279 | 346,710 |
| | VNS | 33.88 | 310,876 | 1,648,990 |
| | VNS + Aug. | 33.96 | 310,126 | 1,648,990 |
| | VNS + Enh. | 33.14 | 1,968,606 | 10,778,083 |
| | VNS + Enh. + Aug. | 33.10 | 1,874,641 | 10,778,083 |
| | Best-known solutions | 32.17 | – | – |
| | GA—activity list | 36.74 | – | 5000 |

were improved are cited in ftp://ftp.bwl.uni-kiel.de/pub/operations-research/psplib/HTML/datasm.html.

## 6. Conclusion

An effective solution scheme for the RCPSP has been presented. In addition to the use of VNS to explore the solution space, the effectiveness of the scheme is due to progressively reducing the solution space by repeatedly improving both lower and upper bounds, as well as by discovering additional valid precedences to augment the existing set.

The problem has been coded and the solution space explored through the use of valid sequences of activities. The enhanced move strategy developed in this work has contributed much to the efficacy of solution space exploration.

The ideas and techniques developed in this work should be useful also in the context of the various extensions of the RCPSP, such as multi-mode project scheduling, multi-project scheduling and project scheduling with resource-time interactions. This is also true for other problems characterised by the presence of precedence constraints such as the line-balancing problem.

## Acknowledgements

The authors are indebted to anonymous referees for useful comments and suggestions.

## References

[1] R. Alvarez-Valdés, J.M. Tamarit, Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis, in: R. Slowinski, J. Węglarz (Eds.), Advances in Project Scheduling, Elsevier, Amsterdam, 1989.

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

[3] P. Brucker, A. Drexl, R. Mohring, K. Neumann, E. Pesch, Resource-constrained project scheduling: Notation, classification, models and methods, European Journal of Operational Research 112 (1) (1999) 3–41.

[4] E. Davis, J. Patterson, A comparison of heuristic and optimum solutions in resource-constrained project scheduling, Management Science 21 (1975) 944–955.

[5] S. Elmaghrabi, Activity Networks: Project Planning and Control by Network Models, Wiley, New York, 1977.

[6] P. Hansen, N. Mladenović, An introduction to variable neighbourhood search, in: S. Voss, et al. (Eds.), Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization, Kluwer, Dordrecht, 1999, pp. 433–458.

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

[8] S. Hartmann, A competitive genetic algorithm for resource-constrained project scheduling, Naval Research Logistics 45 (7) (1998) 733–750

[9] W. Herroelen, E. Demeulemeester, B. De Reyck, A classification scheme for project scheduling, in: J. Węglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer, Dordrecht, 1999.

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

[11] Y.D. Kim, A backward approach in list scheduling algorithms for multi-machine tardiness problems, Computers and Operations Research 22 (3) (1995) 307–319.

[12] R. Klein, A. Scholl, Computing lower bounds by destructive improvement: An application to resource-constrained project scheduling, European Journal of Operational Research 112 (1999) 322–346.

[13] R. Kolisch, S. Hartmann, Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis, in: J. Węglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer, Dordrecht, 1999.

[14] R. Kolisch, C. Schwindt, A. Sprecher, Benchmark instances for project scheduling problems, in: J. Węglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer, Dordrecht, 1999.

[15] R. Kolisch, A. Sprecher, PSPLIB—A project scheduling problem library, European Journal of Operational Research 96 (1997) 205–216.

[16] R. Kolisch, Project scheduling under resource constraints—Efficient heuristics for several problem classes, Physica, Heidelberg, 1995.

[17] R. Kolisch, Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation, European Journal of Operational Research 90 (1996) 320–333.

[18] N. Mladenović, P. Hansen, Variable neighbourhood search, Computers and Operations Research 24 (11) (1997) 1097–1100.

[19] K. Naphade, S. Wu, R. Storer, Problem space search algorithms for resource-constrained project scheduling, Annals of OR 70 (1997) 307–326.

[20] J.H. Patterson, A comparison of exact approaches for solving the multiple constrained resource project scheduling problem, Management Science 30 (1984) 854–867.
[21] A.A.B. Pritsker, W.D. Watters, P.M. Wolfe, Multiproject scheduling with limited resources: A zero–one programming approach, Management Science 16 (1969) 93–108.
[22] A. Sprecher, R. Kolisch, A. Drexl, Semi-active, active and nondelay schedules for the resource-constrained project scheduling problem, European Journal of Operational Research 80 (1995) 94–102.
[23] J. Węglarz (Ed.), Project Scheduling: Recent Models, Algorithms and Applications, Kluwer, Dordrecht, 1999.