UNIVERSITEIT GENT

FACULTEIT ECONOMIE EN BEDRIJFSKUNDE

HOVENIERSBERG 24
B-9000 GENT
Tel. : 32 - (0)9 – 264.34.61
Fax. : 32 - (0)9 – 264.35.92

WORKING PAPER

Meta-heuristic resource-constrained project scheduling: solution space restrictions and neighbourhood extensions

Dieter Debels [1]
Mario Vanhoucke [2]

May 2006

2006/387

[1] Faculty of Economics and Business Administration, Ghent University, Gent, Belgium
[2] Faculty of Economics and Business Administration, Ghent University, Gent, Belgium
Operations & Technology Management Centre, Vlerick Leuven Gent Management School, Gent, Belgium
dieter.debels@ugent.be • mario.vanhoucke@ugent.be

D/2006/7012/32

ABSTRACT

The resource-constrained project scheduling problem (RCPSP) has been extensively investigated during the past decades. Due to its strongly NP-hard status and the need for solving large realistic project instances, the recent focus has shifted from exact optimisation procedures to (meta-) heuristic approaches. In this paper, we extend some existing state-of-the-art RCPSP procedures in two ways. First, we extensively test a decomposition approach that splits problem instances into smaller sub-problems to be solved with an (exact or heuristic) procedure, and re-incorporates the obtained solutions for the sub-problems into the solution of the main problem, possibly leading to an overall better solution. Second, we study the influence of an extended neighbourhood search on the performance of a meta-heuristic procedure. Computational results reveal that both techniques are valuable extensions and lead to improved results.

Keywords: *Resource-constrained project scheduling; Problem decomposition; Meta-heuristics*

Introduction

The RCPSP (problem $m,1|cpm|C_{max}$ using the classification scheme of Herroelen, Demeulemeester and De Reyck (1998)) can be stated as follows. In a project network in AoN format $G(N,A)$, we have a set of nodes N, and a set of pairs A, representing the direct precedence relations. The set N contains n activities, numbered from 1 to n ($|N| = n$). Furthermore, we have a set of resources R, and for each resource type $k \in R$, there is a constant availability a_k throughout the project horizon. Each activity $i \in N$ has a deterministic duration $d_i \in \mathbb{N}$ and requires $r_{ik} \in \mathbb{N}$ units of resource type k. We assume that $r_{ik} \leq a_k$ for $i \in N$ and $k \in R$. The dummy start and end activities 1 and n have zero duration and zero resource usage. A schedule S is defined by an n-vector of starting times $s(S) = (s_0, ..., s_n)$, which implies an n-vector of finishing times $f(S)$ where $f_i = s_i + d_i$, $\forall i \in N$. A schedule is said to be feasible if it is non-pre-emptive and if the precedence and resource constraints are satisfied. The objective of the RCPSP is to find a feasible schedule that minimizes the schedule makespan f_n.

The research on the RCPSP has been investigated extensively over the last few decades, and reviews can be found in Brucker et al. (1999), Herroelen, De Reyck and Demeulemeester (1998), Icmeli, Erenguc and Zappe (1993), Kolisch and Padman (2001) and Özdamar and Ulusoy (1995). Due to the need for solving larger project instances, the recent research focus has shifted to the development of meta-heuristics. Kolisch and Hartmann (1999) present a classification and performance evaluation of existing heuristic procedures. An excellent review paper by Kolisch and Hartmann (2004) discusses the different meta-heuristics for the RCPSP and is an update of the previously published paper of Hartmann and Kolisch (2000). This research revealed that very diverse meta-heuristic techniques have been applied to the RCPSP, but the best performing procedures all have some characteristics in common. They often make use of a population-based meta-heuristic framework, flavoured with some

problem-specific components such as the incorporation of the topological order condition (schedule representation) and the iterative forward/backward scheduling technique (local search method) (Li and Willis, 1992; Valls, Ballestin and Quintanilla, 2005).

In this paper, we extend three population-based meta-heuristics for the *resource-constrained project scheduling problem* as follows:

- The decomposition of a problem instance to gain efficiency in the search process: Splitting problem instances into smaller sub-problems in order to *restrict the search process* has been investigated by various authors (Mausser and Lawrence (1997), Sprecher (2002), Palpant, Artigues and Michelon (2004), Debels and Vanhoucke (2006b)). In the current manuscript, we rely on the principles of the decomposition approach of Debels and Vanhoucke (2006b) who have developed a decomposition-based genetic algorithm for the RCPSP. We investigate the incorporation of an exact tree search procedure to solve sub-problems and test its impact on the solution quality of the project schedule.
- The investigation of different cross-over operators to improve the neighbourhood search: We investigate the impact of a combination of three well-known neighbourhood search principles into an *extended neighbourhood search* on the solution quality of the meta-heuristic.

We believe that our two extensions provide insights to develop procedures that report high-quality solutions under a wide range of test design assumptions (various data instances, stop criteria, etc…). The three selected population based meta-heuristics, a genetic algorithm (GA), a scatter search (SS) and an electromagnetism (EM) procedure, have been successfully implemented for the RCPSP and have been proven to outperform many other state-of-the-art procedures (Debels and Vanhoucke, 2006b). In this paper, we show that the incorporation of our two extensions leads to better results than the use of the individual procedures, and therefore seems to be a promising area for future research in project scheduling.

The outline of this paper is as follows. In section 1 we briefly review the building blocks of the three population-based meta-heuristics under study, i.e. an electromagnetism approach, a scatter search approach and a genetic algorithm. In section 2 we rely on these three techniques to investigate the impact of the decomposition based approach on the solution quality. Section 3 investigates the influence of the variable neighbourhood search on the quality of the meta-heuristic procedure. Finally, we draw conclusions and report areas for future research in section 4.

1 Three population-based heuristics for the RCPSP

In this section, we review the building blocks of our three meta-heuristics under study. These heuristic procedures are among the best performing heuristic procedures for the RCPSP, thanks to the unique representation of a schedule and the use of a very efficient and effective local search procedure (Debels et al., 2006). These components are described shortly in section 1.1. Afterwards, we outline the main characteristics of each individual procedure in section 1.2.

1.1 Common characteristics of the three meta-heuristics

The representation and evaluation of a schedule determine the backbone of a meta-heuristic for the RCPSP. The *schedule representation* serves as an encoding of a schedule, which needs to be transformed into a schedule by means of a *schedule generation scheme* (SGS). For both the representation and generation of a schedule, various approaches exist, as described in section 1.1.1. The local search procedure that has been implemented in all the heuristics relies on forward and backward calculations of Li and Willis (1992), and is the topic of section 1.1.2.

1.1.1 Representation and generation of a schedule

Kolisch and Hartmann (1999) distinguish five different schedule representations, but the *activity-list* (AL) representation and the *random-key* (RK) representation are the most widely-spread. In both representations, a priority structure between the activities is embedded. The AL representation obtains this structure by making use of a sequence of the activities. The position of an activity in this sequence determines its relative priority versus the other activities. The RK representation that is utilized in our procedures uses a vector $\boldsymbol{x} \in \mathrm{IR}^n$ such that x_i denotes the priority value of activity i. Ideally, a solution (i.e. a schedule) representation should meet the following five conditions (Palmer and Kershenbaum, 1994):

(i) The transformation between solutions should be computationally fast
(ii) For each solution in the original space, there is a solution in the encoded space
(iii) Each encoded solution corresponds to one feasible solution in the original space
(iv) All solutions in the original space should be represented by the same number of encoded solutions
(v) Small changes in the encoded solution should result in small changes in the solution itself

Neither the AL nor the RK fulfil these requirements as they can both have many different representations for one single schedule. The so-called topological order (TO) condition of Valls, Quintanilla and Ballestin (2003) overcomes this problem and uses a unique, standardized form of the RK representation. The TO condition implies that for all activities i and j for which $s_i(S) < s_j(S)$, activity i should have a higher priority than activity j. In order to incorporate the TO condition, we use the so-called SRK representation of Debels et al. (2006). More precisely, we first rank the activities according to their start times in the schedule, and then replace their priority values by the place in the ranking. By using the SRK representation, each schedule is uniquely associated with an RK-vector. However, after a move during the neighbourhood search, the newly generated priority vectors are no longer in SRK format. Hence, each new priority vector will be transformed into the SRK-form, while at the same time evaluating the associated objective function value. Debels and Vanhoucke (2006a) have explicitly shown the beneficial effect of introducing the SRK representation.

1.1.2 Local search method

The local search procedure that we have implemented in all our heuristics makes use of the iterative forward/backward scheduling technique (Li and Willis, 1992) in which both left-justified and right-justified schedules are used. A left-justified schedule is obtained by iteratively scheduling precedence-feasible activities forwards. To get a right-justified schedule, the precedence-relations should be reversed such that precedence-feasible activities can be scheduled backwards. The iterative forward/backward scheduling technique iteratively transforms a left-justified schedule into a right-justified schedule and a right-justified schedule into a left-justified schedule. We use the iterative forward/backward technique as a local search method by using starting times (finishing times) of a right-justified (left-justified) schedule as a priority-rule to build a left-justified (right-justified) schedule. Debels et al. (2006) have shown that this local search method can only lead to schedule-improvements.

1.2 Description of the three meta-heuristics

In this section, we review the building blocks of the three meta-heuristics under study: an EM algorithm (Debels and Vanhoucke, 2006a), a SS algorithm (Debels et al., 2006) and a genetic algorithm (Debels and Vanhoucke, 2006b).

1.2.1 The EM algorithm

Electromagnetism has been introduced by Birbil and Fang (2003) as an optimization heuristic for global optimization problems, i.e. the minimization of non-linear functions. In this technique, each solution point (i.e. an SRK vector representing a schedule) is considered as a point in a multi-dimensional solution space with a certain *charge*. This charge is related to the objective function values associated with all the solution points. An initial population is created, in which each solution point will exert attraction or repulsion on other points by computing the forces of each point. The magnitude of the forces is related to the product of the charges and inversely related to the distance between the points and obeys the well-known law of Coulomb. The application of the forces results in new solution points, such that inferior solution points will prevent a move in their direction by repelling other solution points in the population, and attractive points will facilitate moves in their direction. Consequently, the main difference with existing methods is that the moves are governed by forces that obey the rules of electromagnetism. Birbil and Fang (2003) provide the following generic pseudo-code for the EM algorithm:

```
Algorithm EM
  1. Construct a pool of arbitrary solutions
  While (stop criterion not met) do
    2. Local search
    3. Compute forces
    4. Apply forces
  Endwhile
```

This EM technique has been successfully implemented by Debels and Vanhoucke (2006a) for the RCPSP. They show that the procedure performs reasonably well compared to the best state-of-the-art heuristics in the literature. More precisely, the solution quality of this procedure lies close to the best performing meta-heuristics in literature but cannot outperform them. However, hybridising other meta-heuristic techniques with ideas borrowed from EM might certainly lead to new improved solution procedures.

1.2.2 The SS algorithm

Scatter search is a population-based method that has been applied successfully to various optimisation problems. Scatter search contrasts with other evolutionary procedures by providing strategic designs where other approaches resort to randomisation. For a general introduction to scatter search, we refer to Glover, Laguna and Marti (2000) and Martí, Laguna and Glover (2004). The pseudo code of SS can be displayed as follows:

```
Algorithm SS
  1. Construct a pool of arbitrary solutions
  2. Construct an initial reference set from the pool
  While (stop criterion not met) do
    3. Generate subsets
    4. Create a pool of new trial solutions by applying a
       solution combination method to each subset
    5. Update the reference set
  Endwhile
```

After the construction of a large pool of randomly generated solutions, the algorithm relies on a 'two-tier' design to construct the reference set. This set contains a set with high-quality solutions (Set1) and a set with diverse solutions (Set2). The subsets are generated by combining solutions of Set1 and by combining solutions of Set1 and Set2.

1. All pairs in Set1: From each pair, two children are produced using a standard two-point crossover and are added to the pool. This crossover operator is similar to the one taken by Debels et al. (2006).
2. Elements from Set1 × Set2: In Debels et al. (2006), the children produced from one element of Set1 and one element of Set2 are generated by means of EM.

We rely on a static update of the reference set, by adding elements to Set1 and Set2 after the complete generation of each new pool. Comparative computational results indicated that this RCPSP-heuristic belongs to the top-ranked procedures from literature.

1.2.3 The GA algorithm

The GA technique simulates the evolution of living beings and incorporates the 'survival of the fittest' principle to solve complex optimization problems (Holland, 1975). In a GA, processes loosely based on natural selection, crossover and mutation are repeatedly applied to a population that represents potential solutions. In recent decades, operations research literature has been overwhelmed with

genetic algorithms for different project scheduling problems. The GA used in this paper deviates somewhat from a classical GA approach, as shown in the pseudo-code below:

```
Algorithm GA
  1. Construct a pool of arbitrary left-justified solutions (LJ)
  While (stop criterion not met) do
    2. Select parents from LJ set
    3. Generate children by the cross-over operator
    4. Update the RJ set
    5. Select parents from RJ set
    6. Generate children by the cross-over operator
    7. Update the LJ set
  Endwhile
```

Contrarily to a conventional GA, this meta-heuristic uses two separate populations containing left-justified (LJ) and right-justified (RJ) schedules. The procedure starts with the generation of an initial left population, followed by an iterative process that continues until the stop criterion is met. The iterative process consecutively adapts the left-justified (right-justified) population elements in a sequence. The right (left) population is updated by feeding it with combinations of population elements taken from the left (right) population. In doing so, the left (right) population-elements are transformed in right-justified (left-justified) schedules. This unique approach allows to fully exploit the advantages of the local search procedure described in section 1.1.2. To combine population elements, we use a two-point cross-over operation that relies on the calculation of the *Total Resource Utilization* (TRU) (Debels and Vanhoucke, 2006b) and is an extended idea of the multi-point peak crossover operator of Valls et al. (2002). Basically, the TRU aims at replacing (weak) sub-schedules with a low resource utilization of the father by better corresponding sub-schedules of the mother. As computational results have shown, this GA procedure outperforms all other RCPSP-heuristics from literature.

2 Restricting the solution space

In this section, we investigate the impact of a solution space restriction by iteratively solving decomposed sub-problems of the main problem under study. A decomposition approach for the RCPSP has been proposed by Mausser and Lawrence (1997), Sprecher (2002), Palpant, Artigues and Michelon (2004) and Debels and Vanhoucke (2006b). In the current manuscript, we rely on the decomposition approach of Debels and Vanhoucke (2006b), who developed a decomposition-based genetic algorithm for the RCPSP. Their algorithm initially searches for a feasible start solution for the RCPSP problem instance under study, and iteratively splits each problem instance into smaller sub-problems. The search is then continued on the sub-problems, and the resulting sub-schedules can be reincorporated in the schedule of the main problem, possibly leading to an overall better solution.

Although Debels and Vanhoucke (2006b) used a genetic algorithm for solving both the main problem (initial start solution) and the sub-problems, their general approach can be extended to any solution procedure for the RCPSP. The general pseudo-code of their decomposition approach, which will be used throughout the remainder of this paper, can be summarized as follows:

```
Algorithm decomposition_based_heuristic
  1. Run an algorithm on the main-problem until a stop criterion is met
  For i = 1 to iter
    2. Construct the i^th subproblem
    3. Run an algorithm on the subproblem until
       a stop criterion is met
    4. Embed the subproblem into the main problem
```

The algorithm iteratively solves a different subpart of the main problem (the number of iterations equals '*iter*'). More precisely, the decomposition based heuristic (DBH) iteratively focuses on a later part of the schedule such that it has a small overlap with the part of the sub-schedule considered at the previous iteration. In doing so, we guarantee that each part of the main schedule is the subject of an intensive search process in the DBH. We rely on a pre-specified number of decomposed sub-problems '*iter*' as an input-parameter, and calculate the size of the sub-problems inversely related to the value of '*iter*'.

In order to investigate the overall performance of the DBH, we use the three previously described meta-heuristics (EM, GA and SS) to solve each main problem instance. In section 2.1, we solve each sub-problem with the same meta-heuristic used for the initial feasible start solution of the main problem. In section 2.2, we rely on an exact approach to find solutions for the sub-problems. Although the effectiveness of exact solution approaches is often restricted to relatively small instances, the decomposition approach allows a problem size reduction into manageable sub-problems.

We have coded all procedures in Visual C++ 6.0 and performed computational tests on an Acer Travelmate 634LC with a Pentium IV 1.8 GHz processor using two datasets. The first one is the well-known PSPLIB testset (Kolisch and Sprecher, 1997), which we use to compare our procedure with other existing procedures from literature. The second dataset RG300 contains 480 large problem instances and has been generated by Debels and Vanhoucke (2006b) using *RanGen* (Demeulemeester, Vanhoucke and Herroelen, 2003). Each instance contains 300 activities and 4 resources. The order-strength is set at 0.25, 0.50 or 0.75, resource usage at 1, 2, 3 or 4 and the resource-constrainedness at 0.2, 0.4, 0.6 or 0.8. Using 10 instances for each problem class, we obtain a problem set with 480 network instances.

2.1 The use of meta-heuristics for the DBH

In table 1, we display the computational results for the DBH, using EM, SS and GA both for the main problems and the sub-problems. The last two columns compare results without (*iter* = 0) and with (*iter* ≥ 0) decomposition and measure the contribution of the DBH on the performance of the meta-heuristic procedures. We tested all procedures without decomposition (*iter* = 0) under a strict stop condition of 1,000; 5,000 and 50,000 schedules, as proposed by Hartmann and Kolisch (2000). However, the decomposition-based procedures (*iter* ≥ 0) are tested differently. Using a strict schedule limit stop condition would be unfair since the construction of sub-schedules requires less CPU-time than the construction of a complete schedule. Therefore, we use a time-equivalent approach of Debels and Vanhoucke (2006b) which truncated the procedures after a pre-specified time limit that is equal to the CPU-time needed to solve each corresponding problem instance without decomposition. Hence, the rows labelled with "Avg.CPU" display the average CPU-time to solve the problem instances, and are equal for the procedures with (*iter* ≥ 0) and without decomposition (*iter* = 0). The rows labelled with "Avg.Dev.Lb" display the average deviation from the optimal solution (for J30) or from the critical path based lower bound (for J60, J90, J120 and RG300). The rows labelled with "*opt_iter*" display the optimal value of *iter* or the number of sub-problems during decomposition. Finally, the rows "b_1 / b_2" (for SS) and "*popsize*" (for EM and GA) report the best-found values for the population size parameters.

The results of table 1 can be summarized as follows. First, the table reveals that the GA outperforms SS and SS outperforms EM, both in terms of computation time and schedule quality. As an example, the GA results for the J90 instances with 5,000 schedules (10.35% within 0.173s) already outperform the EM results with 50,000 schedules (10.45% within 2.45s). Second, the decomposition of problem instances is beneficial for all meta-heuristic procedures and its positive impact increases for large problem instances and small stop conditions. Indeed, the optimal number of iterations increases when problem size increases, and the largest values can be found for the RG300 instances under a stop criterion of 1,000 generated schedules. Consequently, problem decomposition allows to find better solutions for large-sized problem instances in a very quick way. Finally, the results reveal that the optimal population size decreases as the value for *opt_iter* goes up. Indeed, larger *iter* values result in more sub-problems that need to be solved within the same total stop criterion, and hence, the number of schedules in each iteration needs to be decreased.

			iter = 0			*iter* ≥ 0		
			1,000	**5,000**	**50,000**	**1,000**	**5,000**	**50,000**
J30	EM	Avg.CPU	0.01s	0.06s	0.86s	0.01s	0.06s	0.86s
		Avg.Dev.Lb	0.24%	0.12%	0.02%	0.24%	0.12%	0.02%
		opt_iter	0	0	0	0	0	0
		popsize	14	25	100	14	25	100
	SS	Avg.CPU	0.01s	0.07s	0.69s	0.01s	0.07s	0.69s
		Avg.Dev.Lb	0.23%	0.07%	0.01%	0.23%	0.07%	0.01%
		opt_iter	0	0	0	0	0	0
		b_1 / b_2	6 / 4	10 / 3	28 / 3	6 / 4	10 / 3	28 / 3
	GA	Avg.CPU	0.01s	0.06s	0.52s	0.012s	0.06s	0.52s
		Avg.Dev.Lb	0.15%	0.05%	0.02%	0.12%	0.05%	0.02%
		opt_iter	0	0	0	2	0	0
		popsize	45	100	480	42	100	480
J60	EM	Avg.CPU	0.03s	0.13s	1.52s	0.03s	0.13s	1.52s
		Avg.Dev.Lb	11.89%	11.29%	10.98%	11.77%	11.21%	10.96%
		opt_iter	0	0	0	2	2	2
		popsize	8	25	40	8	18	25
	SS	Avg.CPU	0.03s	0.12s	1.28s	0.03s	0.12s	1.28s
		Avg.Dev.Lb	11.68%	11.08%	10.68%	11.62%	11.08%	10.68%
		opt_iter	0	0	0	2	0	0
		b_1 / b_2	4 / 4	10 / 7	26 / 16	3 / 4	10 / 7	26 / 16
	GA	Avg.CPU	0.02s	0.11s	1.11s	0.02s	0.11s	1.11s
		Avg.Dev.Lb	11.45%	10.95%	10.68%	11.31%	10.95%	10.68%
		opt_iter	0	0	0	2	0	0
		popsize	30	90	480	25	90	480
J90	EM	Avg.CPU	0.05s	0.19s	2.45s	0.05s	0.19s	2.45s
		Avg.Dev.Lb	11.52%	10.89%	10.45%	11.28%	10.68%	10.36%
		opt_iter	0	0	0	3	3	3
		popsize	12	25	40	12	20	35
	SS	Avg.CPU	0.04s	0.19s	2.00s	0.04s	0.19s	2.00s
		Avg.Dev.Lb	11.27%	10.57%	10.05%	11.17%	10.54%	10.05%
		opt_iter	0	0	0	2	2	0
		b_1 / b_2	5 / 2	8 / 5	20 / 14	5 / 2	6 / 4	20 / 14
	GA	Avg.CPU	0.04s	0.17s	1.82s	0.04s	0.17s	1.82s
		Avg.Dev.Lb	10.99%	10.35%	9.90%	10.80%	10.35%	9.90%
		opt_iter	0	0	0	2	0	0
		popsize	20	75	440	15	75	440
J120	EM	Avg.CPU	0.07s	0.34s	3.85s	0.07s	0.34s	3.85s
		Avg.Dev.Lb	35.91%	33.54%	32.67%	35.19%	33.33%	32.35%
		opt_iter	0	0	0	3	3	3
		popsize	10	20	40	9	20	40
	SS	Avg.CPU	0.06s	0.31s	3.29s	0.06s	0.31s	3.29s
		Avg.Dev.Lb	34.90%	32.91%	31.45%	34.71%	32.79%	31.44%
		opt_iter	0	0	0	3	3	2
		b_1 / b_2	4 / 2	8 / 5	26 / 12	4 / 2	7 / 5	22 / 12
	GA	Avg.CPU	0.06s	0.27s	3.00s	0.06s	0.27s	3.00s
		Avg.Dev.Lb	34.19%	32.34%	30.82%	33.55%	32.18%	30.69%
		opt_iter	0	0	0	3	3	3
		popsize	20	75	360	18	45	240
RG300	EM	Avg.CPU	0.53s	2.68s	28.83s	0.53s	2.68s	28.83s
		Avg.Dev.Lb	835.55%	828.46%	821.87%	830.34%	822.86%	818.90%
		opt_iter	0	0	0	6	4	4
		popsize	12	14	40	12	12	25
	SS	Avg.CPU	0.47s	2.36s	23.51s	0.47s	2.36s	23.51s
		Avg.Dev.Lb	832.20%	824.85%	817.75%	828.35%	820.69%	815.83%
		opt_iter	0	0	0	5	5	5
		b_1 / b_2	4 / 2	7 / 5	26 / 12	4 / 2	7 / 5	26 / 12
	GA	Avg.CPU	0.34s	1.63s	16.36s	0.34s	1.63s	16.36s
		Avg.Dev.Lb	830.02%	821.80%	812.97%	824.60%	817.36%	809.93%
		opt_iter	0	0	0	8	7	5
		popsize	16	45	280	16	40	260

Table 1. The impact of the DBH on GA, SS and EM

2.2 The use of an exact procedure for the DBH

The use of exact algorithms to solve the RCPSP is often limited to relatively small problem instances. The DBH, however, splits problem instances in smaller sized sub-problems, which suit better to be solved by exact algorithms. In our approach, we rely on a modified version of the branch-and-bound algorithm of Demeulemeester and Herroelen (1992), since this procedure is among the best performing procedures in literature and capable to solve the J30 instance set to optimality in a limited time. We performed experiments on the J120 dataset and results can be found in table 2.

In order to get insights in the performance of the exact solution procedure to solve the sub-problems of the DBH, we compare this solution approach (further referred to as DBH-exact) with the decomposition approach of section 2.1 (further referred to as DBH-heur). However, due to the incorporation of exact code, we are no longer able to use exact time limits as a stop condition for the DBH-exact approach. Indeed, it is only possible to impose a strict stop condition on the first step of the DBH (the initial start solution of the main problem which is solved heuristically), but we have no control on the CPU-time needed to solve the sub-problems to optimality. Therefore, we run our experiments as follows. We run the GA, SS or EM within a stop criterion of 0.1 seconds to generate an initial start solution for the main problem. Afterwards, the exact algorithm is used to solve the sub-problems and we truncate this procedure after 0.15 seconds if no optimal solution has been found. In order to allow a fair comparison, we run the DBH-heur algorithm within the same computational time needed to solve the instances with the DBH-exact approach. The results for both approaches are summarized in table 2. The column labelled with "Avg.CPU" reports the average CPU-time needed to solve an instance by both decomposition approaches. We tested different sub-problem decompositions for the DBH-exact approach, as given in the column labelled with "*iter*" (i.e. 6, 8, 10, 12 and 14). For the DBH-heur approach, we have fine-tuned the value of *iter* as shown in the column labelled with "*opt_iter*". Similar to table 1, we use the average deviation from the critical path based lower bound in the columns "Avg.Dev.Lb" as a measure of solution quality.

The results confirm the previously found results that the GA outperforms both other procedures. Moreover, the table clearly illustrates that the computational effort of the exact procedure restricts the DBH-exact approach to focus on small sub-problems. Hence, the optimal number of iterations (*opt_iter*) for the DBH-heur approach is always lower than the value of *iter* for the DBH-exact approach. The results reveal a U-shaped relation between *iter* and the average deviation from the lower bound, with the best performance obtained at 8 iterations for EM and SS, and 10 iterations for GA. On the one hand, large values for *iter* lead to inferior results due to the restrictive character of the search process (in this case, only small sub-problems have been solved to optimality). On the other hand, small *iter* values result in large sub-problems which can not be solved to optimality within an

acceptable time, and hence, the exact search process needs to be truncated in an early stage, leading to a low solution quality. Consequently, the average CPU-time is also inversely related to the number of iterations. In general, the table reveals that the incorporation of exact code does not lead to improvements compared to the DBH-heur approach.

	Avg.CPU	DBH-exact		DBH-heur	
		iter	Avg.Dev.Lb	*opt_iter*	Avg.Dev.Lb
GA	0.160s	14	33.03%	4	32.71%
	0.236s	12	32.97%	4	32.43%
	0.354s	10	32.95%	4	32.11%
	0.456s	8	33.00%	3	31.94%
	0.510s	6	33.04%	3	31.95%
SS	0.167s	14	33.35%	3	33.27%
	0.243s	12	33.26%	3	33.03%
	0.352s	10	33.19%	3	32.70%
	0.446s	8	33.18%	3	32.56%
	0.503s	6	33.31%	2	32.49%
EM	0.147s	14	33.98%	3	34.25%
	0.225s	12	33.87%	3	33.69%
	0.312s	10	33.78%	3	33.29%
	0.371s	8	33.78%	3	33.20%
	0.378s	6	33.94%	3	33.16%

Table 2. Results for the DBH with an exact solution procedure

To improve the performance of the DBH-exact approach, we have also performed experiments in which network information has been incorporated in the sub-problem generation. More precisely, we rely on the research of Herroelen and De Reyck (1999) who have investigated the complexity of the RCPSP and tested the impact of two indicators, the order strength (*OS*) (Mastor, 1970), and the resource-constrainedness (*RC*) (Patterson, 1976), on the problem complexity. For RCPSP-instances they have shown a hard-easy phase transition for the *OS* and an easy-hard-easy phase transition for the *RC*. We have used this information to control our search process and influenced the size of the sub-problem to networks with low values for *OS* ($OS \leq 0.19$) and values for *RC* between 0.14 and 0.35. In doing so, sub-problems can be optimally solved in a very small amount of computational time. As a result, table 3 reveals that the DBH-exact approach under a controlled sub-problem generation has a better performance than the results of table 2. The controlled DBH-exact approach outperforms the DBH-heur approach for EM, and leads to improved results for the SS and GA algorithms when the number of sub-problems is large enough (i.e. for $iter \geq 10$ (SS) and $iter = 14$ (GA)). Consequently, the controlled DBH-exact approach might be a good approach to solve large-sized problem instances that need to be decomposed in small and controlled sub-problems.

	Avg.CPU	DBH-exact		DBH-heur	
		iter	Avg.Dev.Lb	*opt_iter*	Avg.Dev.Lb
GA	0.122s	14	33.05%	4	33.17%
	0.132s	12	33.04%	4	32.88%
	0.142s	10	33.03%	4	32.81%
	0.148s	8	33.03%	4	32.82%
	0.159s	6	33.08%	4	32.76%
SS	0.126s	14	33.39%	3	33.57%
	0.134s	12	33.33%	3	33.51%
	0.142s	10	33.32%	3	33.54%
	0.156s	8	33.36%	3	33.32%
	0.161s	6	33.41%	3	33.26%
EM	0.120s	14	34.04%	3	34.29%
	0.151s	12	34.02%	3	34.02%
	0.164s	10	33.98%	3	33.99%
	0.145s	8	34.01%	3	34.23%
	0.144s	6	34.04%	3	34.23%

Table 3. Results for the DBH with an exact solution procedure and our control mechanism

3 Extending the neighbourhood search

In this section, we investigate the potential of combining different cross-over operators into a variable neighbourhood search on the solution quality of the meta-heuristic procedures. Population-based meta-heuristics are solution procedures that make use of a set of individuals (i.e. the population) that evolves during the search process. During this transformation process, exchange of information between two or more individuals is a matter of degree. The way this information exchange is done depends on the particular features of the meta-heuristic technique. The EM heuristic relies on the use of forces to generate new solutions, the SS technique relies on a solution combination method to combine different schedules and the GA technique makes use of so-called cross-over operators to generate children from a father and a mother solution. In the remainder of this paper, we use the general term 'cross-over operator' to refer to the exchange of information between two or more individuals to generate new solutions. In this section, we study the impact of the choice of an individual crossover operator on the performance of the GA that has been proven to outperform all other procedures (see table 1), and investigate the impact of the combination of multiple crossovers. We selected three different crossover operators from the three best-performing procedures from the literature, namely the *random 2-point crossover* operator (rnd_2-pt, Debels et al. (2006)), the *peak multi-point crossover* operator (multi-pt, of Valls, Ballestin and Quintanilla (2002)) and the *peak 2-point crossover* operator (peak_2-pt, of Debels and Vanhoucke (2006b)).

Table 4 displays the results for the J30, J60 and J120 instances under a stop condition of 1,000, 5,000 and 50,000 generated schedules. The rows labelled with "Avg.Dev.Lb" report the average deviation from the optimal solution (J30) or from the critical path based lower bound (J60 and J120). We tested

various settings for the population size, for which the best value has been reported in the rows labelled with "*popsize*". We have tested the three previously mentioned crossover operators (i.e. rnd_2-pt, peak_2-pt and multi-pt) and combined them into the "combined crossover". The relative importance of each crossover operator that leads to the best results has been displayed for the combined crossover operator in "% rnd_2-pt", "% peak_2-pt" and "% multi-pt".

The results clearly illustrate the beneficial effect of combining the various crossover operators into one algorithm. In all cases, the combined crossover operator outperforms the individual operators. As an example, the average deviation decreases from 32.34% to 32.28% for J120 and 5,000 schedules when combining the three crossover operators of our study. The results also reveal that the relative importance of the multi-pt crossover is rather low (always less than 10%). This crossover operator has a weak performance on the solution quality, and performs worse than the two other operators.

			J30	**J60**	**J120**
1,000	rnd_2-pt	Avg.Dev.Lb	0.17%	11.45%	34.29%
		popsize	46	30	20
	peak_2-pt	Avg.Dev.Lb	0.15%	11.45%	34.19%
		popsize	45	30	20
	multi-pt	Avg.Dev.Lb	0.28%	11.82%	36.15%
		popsize	44	36	20
	combined crossover	Avg.Dev.Lb	0.11%	11.44%	34.16%
		popsize	46	30	18
		% rnd_2-pt	40%	60%	40%
		% peak_2-pt	50%	40%	60%
		% multi-pt	10%	0%	0%
5,000	rnd_2-pt	Avg.Dev.Lb	0.06%	11.00%	32.34%
		popsize	120	71	60
	peak_2-pt	Avg.Dev.Lb	0.04%	10.95%	32.34%
		popsize	100	90	75
	peak multi-point	Avg.Dev.Lb	0.12%	11.19%	33.76%
		popsize	140	110	60
	combined crossover	Avg.Dev.Lb	0.04%	10.94%	32.28%
		popsize	120	80	50
		% rnd_2-pt	0%	40%	50%
		% peak_2-pt	100%	50%	40%
		% multi-pt	0%	10%	10%
50,000	rnd_2-pt	Avg.Dev.Lb	0.02%	10.69%	30.75%
		popsize	370	390	290
	peak_2-pt	Avg.Dev.Lb	0.02%	10.68%	30.82%
		popsize	480	480	360
	peak multi-point	Avg.Dev.Lb	0.03%	10.84%	32.64%
		popsize	410	350	220
	combined crossover	Avg.Dev.Lb	0.02%	10.64%	30.74%
		popsize	400	380	260
		% rnd_2-pt	50%	30%	60%
		% peak_2-pt	50%	60%	40%
		% multi-pt	0%	10%	0%

Table 4. Computational results with three crossover operators on the GA

4 Conclusions

In this paper, we relied on state-of-the-art meta-heuristic procedures to solve the resource-constrained project scheduling problem and investigated two general extensions, as follows:

First, the decomposition of a problem instance to smaller sub-problems based on the principles of the decomposition-based heuristic proposed by Debels and Vanhoucke (2006b) has been extensively investigated. We have extended this approach by using three meta-heuristic procedures and an efficient exact procedure from literature. The results obtained from extensive computational tests are promising. Test designs with large problem instances and low stop conditions show that traditional meta-heuristics have insufficient time to efficiently explore the entire solution space. Hence, embedding these procedures into the decomposition based heuristic overcomes this problem, and leads to improved results. Moreover, the incorporation of exact search algorithms to solve the sub-problems leads to improvements, but the size of the sub-problems needs to be carefully controlled based on well-known complexity indicators from literature. This control mechanism is needed to select sub-problems that are large enough to find improvements for the schedule of the main problem, and small enough to avoid excessive computation times to solve to optimality.

Second, the detailed investigation of different cross-over operators to enlarge the neighbourhood search leads to improved results. To that purpose, we combined three well-known crossover operators from literature in the genetic algorithm. Computational results show that this neighbourhood extension leads to improvements for problem instances truncated after a large amount of time.

8 References

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

Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E. (1999). Resource-constrained project scheduling: notation, classification, models and methods, *European Journal of Operational Research*, 112, 3-41.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M. (2006). A hybrid scatter-search/electromagnetism meta-heuristic for the resource-constrained project scheduling problem, *European Journal of Operational Research*, 169 (3), 638-653.

Debels, D., Vanhoucke, M. (2006a). An electromagnetism meta-heuristic for the resource-constrained project scheduling problem, *Lecture Notes in Computer Science*, 3871, 259-270.

Debels, D., Vanhoucke, M. (2006b). A decomposition-based heuristic for the resource-constrained project scheduling problem, to appear in *Operations Research*.

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

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

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

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

Herroelen, W., De Reyck, B. (1999), Phase transitions in project scheduling, *Journal of Operational Research Society*, 50, 148-156.

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

Herroelen, W., Demeulemeester, E., De Reyck, B. (1998). A classification scheme for project scheduling. *In: Weglarz, J. (Ed.), Project Scheduling – Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 14, pp. 77-106.

Holland, J.H. (1975). Adaptation in natural and artificial systems. *The University of Michigan Press*, Ann Arbor.

Icmeli, O., Erenguc, S.S., Zappe, C.J. (1993). Project scheduling problems: a survey, *International Journal of Operations and Productions Management*, 13 (11), 80-91.

Kolisch, R., Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. *In: Weglarz, J. (Ed.), Project Scheduling – Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, pp. 147-178.

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

Kolisch, R., Hartmann, S. (2004). Experimental investigation of heuristics for resource-constrained project scheduling: an update, *working paper*, Technical University of Munich.

Kolisch, R., Sprecher, A. (1997). PSPLIB - A project scheduling library, *European Journal of Operational Research*, 96, 205-216.

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

Martí, R., Laguna, M. and Glover, F. (2006). Principles of scatter search. *European Journal of Operational Research*, 169 (2), 359-372.

Mastor, A. (1970). An experimental and comparative evaluation of production line balancing techniques, *Management Science*, 16, 728-746.

Mausser, H.E., Lawrence, S.R. (1997). Exploiting block structure to improve resource-constrained project schedules. *In: Glover, F., Osman, I. and Kelley, J. (Eds.), Metaheuristics 1995: State of the art*, Kluwer, Maryland, 1997.

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

Palmer, C., Kershenbaum, A. (1994). Representing trees in genetic algorithms. *Proceedings of the first IEEE International Conference on Evolutionary Computation*, New York, 379-384.

Palpant, M., Artigues, C., Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131, 237-257.

Patterson, J.H. (1976), "Project scheduling: the effects of problem structure on heuristic scheduling", *Naval Research Logistics*, 23, 95-123.

Sprecher, A. (2002). Network decomposition techniques for resource-constrained project scheduling. *Journal of the Operational Research Society*, 53 (4), 405-414.

Valls, V., Ballestín, F., Quintanilla, S. (2002). A hybrid genetic algorithm for the Resource-constrained project scheduling problem with the peak crossover operator, *Eighth International Workshop on Project Management and Scheduling*, 368-371.

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

Valls, V., Ballestin, F., Quintanilla, S. (2005). Justification and RCPSP: A technique that pays,

European Journal of Operational Research, 165 (2), 375-386.