Vlerick Leuven Gent Management School

the Autonomous Management School of
Ghent University and Katholieke Universiteit Leuven

Vlerick Leuven Gent Working Paper Series 2005/12

A DECOMPOSITION-BASED HEURISTIC FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

DIETER DEBELS

MARIO VANHOUCKE

Mario.Vanhoucke@vlerick.be

D/2005/6482/12

A DECOMPOSITION-BASED HEURISTIC FOR THE RESOURCE-CONSTRAINED PROJECT SCHEDULING PROBLEM

DIETER DEBELS
Faculty of Economics and Business Administration, Ghent University
MARIO VANHOUCKE
Vlerick Leuven Gent Management School

Contact:
Mario Vanhoucke
Vlerick Leuven Gent Management School
Tel: +32 09 210 97 81
Fax: +32 09 210 97 00
Email: Mario.Vanhoucke@vlerick.be

ABSTRACT

In the last few decades the resource-constrained project scheduling problem has become a popular problem type in operations research. However, due to its strongly *NP*-hard status, the effectiveness of exact optimisation procedures is restricted to relatively small instances. In this paper we present a new genetic algorithm (GA) for this problem, able to provide near-optimal heuristic solutions. This GA procedure has been extended by a so-called decomposition-based heuristic (DBH) which iteratively solves subparts of the project. We present computational experiments on two datasets. The first benchmark set is used to illustrate the contribution of both the GA and the DBH. The second set is used to compare the results with current state-of-the-art heuristics, and to show that the procedure is capable of producing consistently good results for challenging problem instances. We illustrate that GA is currently the best performing RCPSP meta-heuristic, and that the DBH further improves the performance of the GA.

1 INTRODUCTION

We study the resource-constrained project scheduling problem (RCPSP), denoted as $m,1|cpm|C_{max}$ using the classification scheme of Herroelen, Demeulemeester and De Reyck (1999). The RCPSP can be stated as follows. In a project network in AoN format $G(\boldsymbol{N},\boldsymbol{A})$, we have a set of nodes $\boldsymbol{N}$ and a set of pairs $\boldsymbol{A}$, representing the direct precedence relations. The set $\boldsymbol{N}$ contains n activities, numbered from 1 to n ($|\boldsymbol{N}| = n$). The set of pairs $\boldsymbol{A}^+$ adds the transitive precedence relations to $\boldsymbol{A}$. Furthermore, we have a set of resources $\boldsymbol{R}$, and for each resource type $k \in \boldsymbol{R}$, there is a constant availability a_k throughout the project horizon. Each activity $i \in \boldsymbol{N}$ has a deterministic duration $d_i \in$ IN and requires $r_{ik} \in$ IN units of resource type k. We assume that $r_{ik} \leq a_k$ for $i \in \boldsymbol{N}$ and $k \in \boldsymbol{R}$. 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 start times $\boldsymbol{s}(S) = (s_0, ..., s_n)$, which implies an n-vector of finish times $\boldsymbol{f}(S)$ where $f_i = s_i + d_i$, $\forall i \in \boldsymbol{N}$. A schedule is said to be feasible if it is nonpreemptive and if the precedence and resource constraints are satisfied. The objective of the RCPSP is to find a feasible schedule that minimizes the project makespan f_n.

The research on the RCPSP has widely expanded 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). Numerous exact solution approaches have been advanced, such as Brucker et al. (1998), Demeulemeester and Herroelen (1992, 1997), Mingozzi et al. (1998) and Sprecher (2000). However, these procedures are only capable to solve relatively simple problem instances. This has motivated researchers to develop heuristic methods for dealing with more challenging RCPSP instances. Hartmann and Kolisch (2000) present a performance evaluation of existing heuristic procedures. An update is given in Kolisch and Hartmann (2004).

The meta-heuristic procedures of Alcaraz and Maroto (2001), Tormos and Lova (2001, 2003a), Valls, Ballestin and Quintanilla (2002 and 2004a), Valls, Quintanilla and Ballestin (2003) and Debels et al. (2004) make 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.

Debels et al. (2004) have used the iterative forward/backward scheduling technique as a local search method and illustrated that, by using start times (finish times) of a right-justified (left-justified) schedule as a priority rule to build a left-justified (right-justified) schedule, only improvements are possible.

In this paper we present a genetic algorithm (GA) for the resource-constrained project scheduling problem. Furthermore, this algorithm will be embedded in a heuristic search procedure (further referred to as the decomposition-based heuristic (DBH)) that iteratively solves subparts of the problem under study. More precisely, the DBH selects a sub-problem from a feasible solution for the main problem under study. The GA is then used to find a high-quality solution for the sub-problem which can be reincorporated in the solution of the main problem.

The outline of this paper is as follows. In section 2 we describe our genetic algorithm approach for the RCPSP in detail. In section 3 we explain the different steps of our Decomposition-Based Heuristic (DBH). Section 4 discusses extensive computational results of the DBH and compares the performance with other "state-of-the-art" heuristics. Section 5 contains overall conclusions and suggestions for future research.

2 A GENETIC ALGORITHM FOR THE RCPSP

2.1 Representation and generation of a schedule

The representation and evaluation of a schedule determine the backbone of a meta-heuristic for the RCPSP. The *schedule representation* can be seen as an encoding of a schedule. In order to decode the schedule representation into a schedule S, identified by $\boldsymbol{s}(S)$ and $\boldsymbol{f}(S)$, a *schedule generation scheme* (SGS) is necessary. For both the representation and generation of a schedule, various approaches exist.

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 the activity in this sequence determines its relative priority versus the other activities. The RK representation, that is utilized in our procedure, uses a vector $\boldsymbol{x} \in \mathrm{IR}^n$ such that x_i denotes the priority value of activity i. Hartmann and Kolisch (2000) concluded from

experimental tests that procedures based on AL representations outperform the other procedures.

However, Debels et al. (2004) recently illustrated that the RK also leads to promising results thanks to the use of the *topological ordering* (TO) notation (Valls, Quintanilla and Ballestin, 2003). The TO-condition for the construction of left-justified schedules 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. We adapted the TO-condition to our settings, since we rely on two types of schedules (left-justified and right-justified). More precisely, our procedure uses right-justified schedules to generate left-justified children and left-justified schedules to generate right-justified children. In order to fully exploit the advantages of the iterative forward/backward scheduling technique, we embed the TO-condition in the RK-representation as follows:

1. $\boldsymbol{x} := \boldsymbol{f}(S)$ if S is a left-justified schedule
2. $\boldsymbol{x} := \boldsymbol{s}(S)$ if S is a right-justified schedule

In section 2.2, we rely on this specific TO-condition to perform the cross-over operations in an effective way.

In the literature there exist also weg different types of SGSs. The serial SGS constructs an active schedule by scheduling each activity one-at-a-time and as soon as possible within the precedence and resource constraints. Alternatively, a parallel SGS could be used. However, Kolisch (1996b) has shown that, contrarily to the serial SGS, the parallel SGS is sometimes unable to reach an optimal solution. This motivated us to use the serial SGS. Our procedure requires two different procedures, "Serial_SGS_left($\boldsymbol{x}$)" and "Serial_SGS_right($\boldsymbol{x}$)", in order to build left-justified and right-justified schedules respectively. These procedures differ in the evaluation of the priority values. If $x_i < x_j$, activity i will have a higher priority than activity j for "Serial_SGS_left($\boldsymbol{x}$)", while the opposite will be true for "Serial_SGS_right($\boldsymbol{x}$)".

Insert Figure 1 & 2 About Here

We introduce the example project depicted in Figure 1, with a single renewable resource type with availability $a_1 = 10$. Table 1 shows the RK-vector $\boldsymbol{x}$. In each column, a priority value is given, belonging to the corresponding activity. The construction of a left-

justified schedule by calling the procedure "Serial_SGS_left(***x***)" and a right-justified schedule by calling "Serial_SGS_right(***x***)", leads to the two schedules of Figure 2.

Insert Table 1 About Here

2.2 Our genetic algorithm

The evolution of living beings motivated Holland (1975) to solve complex optimization problems by using algorithms that simulate biological evolution. This approach gave rise to the technique known as Genetic Algorithm (GA). In a GA processes loosely based on natural selection, crossover and mutation are repeatedly applied to a population that represents potential solutions.

The procedure "Genetic_Algorithm(*C*)" with stop criterion *C*, for which the pseudo-code is given below, incorporates the principles of a GA to obtain a qualitative solution for an RCPSP problem. Contrarily to a conventional genetic algorithm, there are two separate populations: a left population ***PopL*** that contains left-justified schedules and a right population ***PopR*** that contains right-justified schedules. The procedure starts with the generation of an initial left population, followed by an iterative process that continues until the stop criterion *C* is met. The iterative process consecutively adapts the population elements of ***PopL*** and ***PopR***. 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 by means of the procedure "Serial_SGS_right(***x***)" ("Serial_SGS_left(***x***)"). The remainder of this section reveals some further details about the GA.

Procedure Genetic_Algorithm(*C*)

Step 1 Build an initial population ***PopL***

Step 2 while(*C* not satisfied)

2.1 Update_PopR()

2.2 Update_PopL()

Procedure Update_PopR()

1 For [$a = 1$, *popsize*]

1.1 For [$b = 1$, *nr_children*]

1.1.1 Select_Parents(a, ***PopL***)

1.1.2 Crossover to build $\boldsymbol{x_c}$

1.1.3 Diversification of $\boldsymbol{x_c}$

1.1.4 $Child_{a,b}$:= Serial_SGS_right($\boldsymbol{x_c}$)

1.2 Select the best child $Child_a^{best}$

1.3 Replace $PopR_a$ by $Child_a^{best}$

Procedure Update_PopL()

1 For [$a = 1$, *popsize*]

1.1 For [$b = 1$, nr_*children*]

1.1.1 Select_Parents(a, ***PopR***)

1.1.2 Crossover to build $\boldsymbol{x_c}$

1.1.3 Diversification of $\boldsymbol{x_c}$

1.1.4 $Child_{a,b}$:= Serial_SGS_left($\boldsymbol{x_c}$)

1.2 Select the best child $Child_a^{best}$

1.3 Replace $PopL_a$ by $Child_a^{best}$

Build an initial population

We start the genetic algorithm by building an initial population ***PopL*** of *popsize* left-justified schedules. Each population element $PopL_a$ is created by randomly generating a priority vector $\boldsymbol{x_a}$ that is decoded to the left-justified schedule S_a by calling "Serial_SGS_left($\boldsymbol{x}$)". To satisfy the TO-condition, the generated schedules are represented by the priority vectors $PopL_a = \boldsymbol{f}(S_a)$.

Parent Selection

The procedure "Select_Parents(*index*, ***pop***)" selects the couples of population elements for the crossover operator. Two parents are selected from the population ***pop*** as follows. The first parent is the *index*th element of ***pop***, denoted as element pop_{index} (e.g. $PopR_a$ is the a^{th} element of ***PopR***). The second parent is selected using the 2-tournament selection where two population elements from ***pop*** are chosen randomly, and the element with the best objective-function value is selected. Afterwards, we randomly label one element as the father and the other element as the mother.

Crossover operator

After the selection of the parents, a crossover operation combines the RK-representations $\boldsymbol{x_f}$ of the father S_f and $\boldsymbol{x_m}$ of the mother S_m in a new RK-vector $\boldsymbol{x_c}$ of the child S_c. As mentioned before, if S_f and S_m belong to ***PopL* (*PopR*)**, we create a right-justified (left-justified) schedule S_c as a candidate to enter ***PopR* (*PopL***). The combination of the genes of both parents is done by a two-point crossover operator based on a modified version of the peak crossover operator of Valls, Ballestin and Quintanilla (2002) that makes use of the *Resource Utilization Ratio* (RUR). This ratio measures the resource utilization at time unit *t*. Let Active(*t*, *S*) be the set of active activities in schedule *S* at time instant *t*, then RUR is calculated as demonstrated in [1].

$$\mathrm{RUR}(t, S) = \frac{1}{K} \sum_{j \in \mathrm{Active}(t,S)} \sum_{k=1}^{K} \frac{r_{jk}}{a_k} \qquad [1]$$

The RUR allows us to select time intervals for which the resource utilization is high, so-called peaks (Valls, Ballestin and Quintanilla, 2002), and time intervals with low resource utilization. In our procedure we determine one peak, identified by the time interval [$t_1(S)$, $t_2(S)$], representing the crossover points of schedule *S*. To that purpose, we randomly choose the length *l* of the peak between ¼ * makespan(*S*) and ¾ * makespan(*S*) and calculate the *Total Resource Utilization* (TRU) of a sub-schedule with length *l* and start time *t*:

$$\mathrm{TRU}(t, l, S) = \sum_{time=t}^{t+l-1} \mathrm{RUR}(time, S) \qquad [2]$$

The peak with length l is then chosen as the time interval $[t_1(S), t_2(S)]$ by setting $t_1(S)$ at $t \in [0, \text{makespan}(S) - l]$ for which TRU(t, l, S) is maximal and $t_2(S)$ at $t_1(S) + l$. For the remaining intervals $[0, t_1(S)]$ and $[t_2(S), \text{makespan}(S)]$, the average resource utilization will be low.

The two-point crossover operator uses the crossover points $t_1(S_f)$ and $t_2(S_f)$ of the father, to generate the RK-vector of the child as follows:

case 1: If $x_{mi} < t_1(S_f) \Rightarrow x_{ci} := x_{mi} - c$

case 2: If $t_1(S_f) \leq x_{mi} \leq t_2(S_f) \Rightarrow x_{ci} := x_{fi}$

case 3: If $x_{mi} > t_2(S_f) \Rightarrow x_{ci} := x_{mi} + c$

The use of the TO-condition allows the crossover operator to replace the (weak) sub-schedules with a low resource utilization of the father S_f by better corresponding sub-schedules of the mother S_m. In order to prevent that the relative priority structure between the activities of a case will be mixed with priority values of activities of another case, we use c as a large constant.

Insert Figure 3 About Here

Consider Figure 3 that represents an example left-justified schedule S_f of ***PopL***. In order to calculate the crossover points $t_1(S_f)$ and $t_2(S_f)$ of a peak, we first compose the *Resource Utilization Profile*, by calculating the RUR at each time instant t. If l has been chosen randomly to 19, then TRU(t, 19, S_f) has a maximal value of 16.8 when t = 11. Consequently, $t_1(S_f)$ equals 11 and $t_2(S_f)$ equals 30.

Insert Figure 4 About Here

We assume a mother schedule $S_m \in$ ***PopL*** as depicted in Figure 4. Table 2 displays the RK-vectors $\boldsymbol{x_f} = \boldsymbol{f}(S_f)$ and $\boldsymbol{x_m} = \boldsymbol{f}(S_m)$. As 11 and 30 are the crossover points of the father, the RK-vector $\boldsymbol{x_c}$ of the child is calculated as demonstrated in Table 2. The dark coloured activities 5, 6, 7, 9, 10, 14, 15, 17 with $x_{mi} \in [11, 30]$ belong to case 2 and consequently, the priority values x_{fi} are copied in x_{ci}. The activities 11, 12, 13, 16, 18, 19, 20 and 21 with $x_{mi} >$

30 belong to case 3 while the remaining activities 1, 2, 3, 4 and 8 for which $x_{mi} < 11$ belong to case 1. For the light coloured activities of case 1 and case 3, we use the corresponding x_{mi} as priority values in $\boldsymbol{x}_c$. However, a large constant *c* needs to be subtracted for activities of case 1, and added for activities of case 3, to avoid that we mix the priority structures of case 1, case 2 and case 3. In our example, we set *c* at 50.

Insert Table 2 About Here

The corresponding child schedule can be constructed based on the calculated RK-vector $\boldsymbol{x}_c$ of Table 2. Since the parents belong to ***PopL***, we call "Serial_SGS_right($\boldsymbol{x}_c$)" to build the right-justified schedule S_c of Figure 5. As this schedule is a candidate to enter ***PopR***, the RK-vector is set equal to $\boldsymbol{s}(S_c)$.

Insert Figure 5 About Here

Diversification

We implemented a diversification methodology in order to escape from a premature convergence. A lack of diversification leads to a homogeneous population, of which the offspring will be identical to the parents. Our diversification method can be considered as reactive, since it only operates on a child, when it originates from two not mutually diverse parents. To define whether the parents are sufficiently diverse, we need a threshold τ and a distance measure. Since we employ $\boldsymbol{s}(S)$ or $\boldsymbol{f}(S)$, to represent a schedule *S* in ***PopR*** or ***PopL***, we use the sum of the absolute deviations between the priority values of both parents divided by the number of activities as an easy and efficient distance measure. Thus, diversification is desirable if $\tau > \frac{1}{n}\sum_{i=1}^{n}\left|x_{fi} - x_{mi}\right|$ and is exerted on the child $\boldsymbol{x}_c$ by randomly biasing a limited number *nr_div* of priority values by adding a random number between [–*n*, *n*].

Selection mechanism

The selection mechanism determines the way in which the new generation replaces the old generation. In order to make the genetic algorithm successful, the '*survival of the fittest*'-principle should be embedded. Good children should have a higher chance to enter in the new generation than inferior ones in order to improve the quality of the population.

The population ***PopR*** (***PopL***) is fed by children generated from ***PopL*** (***PopR***). In the following, we will explain how we update ***PopR***. The way in which we update ***PopL*** is analoguous. For each element $PopL_a$ we iterate the complete process of building a child *nr_children* times, leading to a set of children ($Child_{a,1}$, …, $Child_{a,nr\_children}$). From this set, we select the child with the lowest makespan $Child_a^{best}$. This schedule will replace the element $PopR_a$, even if this leads to a deterioration of the makespan. However, in order to avoid that we loose high-quality schedules, we do not automatically perform a replacement if $PopR_a$ corresponds to a schedule with the best-found makespan so far. In this case, $PopR_a$ will only be replaced if $Child_a^{best}$ represents a new best-found schedule.

3 THE DECOMPOSITION-BASED HEURISTIC

3.1 Introduction

The genetic algorithm of the previous section can be used as a subroutine in the decomposition-based heuristic (DBH). The DBH runs as follows: an intermediate solution of the main problem (the main schedule) is used as a start base to derive a smaller sub-problem. This sub-problem is solved then with the GA, resulting in an improved sub-schedule. Afterwards, this improved sub-schedule will be reincorporated in the main-schedule, leading to an overall decrease of the total makespan. These three steps can be summarized as follows:

Step 1: Construct sub-problem: This subroutine transforms a schedule S of the main-problem ***P*** into an intermediate schedule S_b and uses that schedule to construct a smaller sub-problem $\boldsymbol{P_{sub}}$ and an initial sub-schedule S_{sub}^{in}.

Step 2: Genetic Algorithm: This subroutine transforms the initial sub-schedule S_{sub}^{in} into an improved sub-schedule S_{sub}^{*}.

Step 3: Merge: This subroutine embeds the improved sub-schedule S_{sub}^{*} into the intermediate schedule S_b, leading to an overall improvement of the total makespan.

In the following sections (3.2, 3.3 and 3.4), we explain each of the three subroutines of the DBH in detail.

3.2 Construct sub-problem

This subroutine uses a right-justified schedule S and a predetermined time interval $[pt_1, pt_2]$ to create an intermediate schedule S_b. This schedule is used to create the sub-problem $\boldsymbol{P_{sub}}$. The details are summarized in the pseudo-code below, using the following symbols. Problem $\boldsymbol{P}$ = G($\boldsymbol{N}$, $\boldsymbol{A}$) with start-schedule S will, based on the transformed intermediate schedule S_b, be reduced to sub-problem $\boldsymbol{P_{sub}}$ = G($\boldsymbol{N_{sub}}$, $\boldsymbol{A_{sub}}$). The set of nodes $\boldsymbol{N_{sub}}$ of sub-problem $\boldsymbol{P_{sub}}$ is further subdivided in the sets $\boldsymbol{B_1}$, $\boldsymbol{B_2}$ and ***Core***. These sets are used in the genetic algorithm approach as will be described in section 3.3.

Procedure Build_sub-problem(S, pt_1, pt_2, $\boldsymbol{P}$)

1 $\boldsymbol{B_1} := \boldsymbol{B_2} := \boldsymbol{Core} := \varnothing$

2 Transform S into S_b

3 Determine $lft = \max\{f_i | s_i < pt_1\}$ and $est = \min\{s_i | f_i > pt_2\}$

4 Construct the sub-problem $\boldsymbol{P_{sub}}$ = G($\boldsymbol{N_{sub}}$, $\boldsymbol{A_{sub}}$)

5 For [i = 2, n-1]

If ($f_i(S_b) > pt_1$ and $s_i(S_b) < pt_2$)

If ($s_i(S_b) \geq lft$ *and* $f_i(S_b) \leq est$) then $\boldsymbol{Core} := \boldsymbol{Core} \cup \{i\}$

If ($s_i(S_b) < lft$) then $\boldsymbol{B_1} := \boldsymbol{B_1} \cup \{i\}$

If ($f_i(S_b) > est$) then $\boldsymbol{B_2} := \boldsymbol{B_2} \cup \{i\}$

Step 1 initializes the sets of activities $\boldsymbol{B_1}$, $\boldsymbol{B_2}$ and ***Core***. In step 2, we transform the schedule S into S_b by using the start times as a priority rule and scheduling all activities that finish before pt_2 forwards. The other activities remain untouched. In this way, the number of active activities within the time interval $[pt_1, pt_2]$ is reduced compared to the schedule S. In step 3, we calculate time window [*lft, est*] with $lft \geq pt_1$ and $est \leq pt_2$. More precisely, *lft* denotes the latest finish time of all activities i in schedule S_b for which $s_i \leq pt_1$ while *est* denotes the earliest start time of all activities i in schedule S_b for which $f_i \geq pt_2$. In step 4 we construct the sub-problem $\boldsymbol{P_{sub}}$ = G($\boldsymbol{N_{sub}}$, $\boldsymbol{A_{sub}}$), defined by the set of activities $\boldsymbol{N_{sub}}$ and the set

of activities $\boldsymbol{A_{sub}}$. $\boldsymbol{N_{sub}}$ contains the dummy activities 0 and n and the activities that are partly or completely active within the time interval [pt_1, pt_2] of schedule S_b. The set of all precedence constraints $\boldsymbol{A^+_{sub}}$ equals $\{(i, j) \in \boldsymbol{A^+} \mid i, j \in \boldsymbol{N_{sub}}\}$. The removal of the transitive arcs from $\boldsymbol{A^+_{sub}}$ results in $\boldsymbol{A_{sub}}$. The durations d_i^{sub} for all activities $i \in \boldsymbol{N_{sub}}$ equal the number of time units each activity i is active within [pt_1, pt_2]. Note that d_i^{sub} corresponds to d_i, unless when i is active at pt_1 or pt_2. In step 5, the set $\boldsymbol{N_{sub}}$ is further subdivided in the sets ***Core***, $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$. ***Core*** contains the activities that are completely active within [*lft*, *est*]. The remaining activities belong to either $\boldsymbol{B_1}$ or $\boldsymbol{B_2}$, depending on whether they are scheduled before *lft* or after *est*.

Insert Figure 6 About Here

In our example we use the right-justified schedule of Figure 2 as *S*. The time interval [pt_1, pt_2] is set randomly at [4, 34]. Figure 6 shows the steps to obtain the resulting sub-problem $\boldsymbol{P_{sub}}$. The intermediate schedule S_b depicted in Figure 6a is the result of forward scheduling of all activities that finish before time instant 34. Consequently, the time interval [*lft*, *est*] equals [8, 29]. Indeed, from the activities that start before $pt_1 = 4$ in S_b, 8 is the finish time of the latest finishing activity 4. 29 corresponds to the start time of the earliest starting activity that finishes later than $pt_2 = 34$. The activities 2, 11, 12, 18, 19 and 20 do not belong to $\boldsymbol{N_{sub}}$, as these activities are never active in S_b within the time interval [pt_1, pt_2] = [4 , 34]. The set [5, 6, 7, 9, 10, 14, 15] determines ***Core***, as these activities are scheduled completely within [*lft*, *est*] = [8, 29]. From the other activities, 3, 4 and 8, that finish before time instant 8, belong to $\boldsymbol{B_1}$, while the activities 13, 16 and 17, that start after time instant 29 belong to $\boldsymbol{B_2}$. Figure 6b displays the network of Figure 1 and the different sets $\boldsymbol{B_1}$, $\boldsymbol{B_2}$, ***Core*** and $\boldsymbol{N_{sub}}$. Figure 6c omits the activities not belonging to $\boldsymbol{N_{sub}}$, and represents the problem $\boldsymbol{P_{sub}}$. For each activity i, the corresponding value for d_i^{sub} is calculated by looking for how many time units i is active within [pt_1, pt_2] = [8, 29] in the schedule S_b. The original duration d_i is added between brackets if this value differs from d_i^{sub}.

3.3 The Genetic Algorithm to solve a sub-problem

After the construction of a sub-problem $\boldsymbol{P_{sub}}$, this sub-problem is the subject of the genetic algorithm described in section 2.2 to find an improved sub-schedule S^*_{sub}. However, some modifications to the schedule generation schemes (i.e. subroutines "Serial_SGS_left($\boldsymbol{x}$)" and "Serial_SGS_right($\boldsymbol{x}$)" of section 2.2) are necessary to make the GA applicable for the DBH. More precisely, the activities of $\boldsymbol{B_1}$, $\boldsymbol{B_2}$ and ***Core*** need to be treated differently. The serial schedule generation scheme to construct a left-justified schedule schedules all activities of $\boldsymbol{N_{sub}}$ based on the random key vector $\boldsymbol{x_{sub}}$, and is applied in four consecutive steps:

Step 1. Schedule the activities of $\boldsymbol{B_1}$ in a sequence determined by the start times of S_b

Step 2. Schedule the activities of ***Core*** based on the RK-values

Step 3. Schedule the activities of $\boldsymbol{B_2}$ in a sequence determined by the start times of S_b and by using the original durations d_i.

Step 4. Replace the durations of $\boldsymbol{B_2}$ by d_i^{sub}

The reason to use the original durations d_i to schedule the activities of $\boldsymbol{B_2}$ is to obtain a schedule that can be incorporated easily into the schedule S_b without incurring pre-emption as will be described in the merge-step of section 3.4. After step 4, we obtain a feasible schedule for the sub-problem.

The construction of a right-justified schedule works oppositely and schedules the activities of $\boldsymbol{B_2}$, ***Core*** and $\boldsymbol{B_1}$ backwards in four consecutive steps. The sequence to schedule $\boldsymbol{B_1}$ and $\boldsymbol{B_2}$ is determined by the finish times of S_b.

Insert Figure 7 About Here

Figure 7 displays a left-justified schedule S_{sub} of $\boldsymbol{P_{sub}}$ based on the RK-vector $\boldsymbol{x_{sub}}$ of Table 3. First, activities 3, 4 and 8 of set $\boldsymbol{B_1}$ are scheduled in a sequence determined by the start times of S_b. Then the dark coloured activities of ***Core*** are scheduled in a sequence based on the RK-vector $\boldsymbol{x_{sub}}$. Finally, activities 13, 16 and 17 are scheduled in a sequence determined by the start times of S_b. To find the earliest time to schedule these three activities, the original durations d_i are used. The makespan of S_{sub} equals 27, which is a reduction of 3 time units compared to the initial schedule embedded in S_b within the time interval [4, 34].

Insert Table 3 About Here

3.4 Merge

This subroutine embeds the improved sub-schedule S^*_{sub} into the intermediate schedule S_b, leading to a new schedule S^* with an overall improvement of the total makespan. To do so, the merge subroutine creates an RK-vector $\boldsymbol{x}^*$ as follows:

1. For $i \in \boldsymbol{Core}$, $x^*_i = pt_1 + f_i(S^*_{sub})$
2. For $i \in \boldsymbol{N} \setminus \boldsymbol{Core}$, $x^*_i = f_i(S_b)$

The vector $\boldsymbol{x}^*$ can be transformed to a right-justified schedule S^*, by applying the schedule generation scheme "Serial_SGS_right($\boldsymbol{x}^*$)".

Insert Table 4 About Here

Let the schedule S_{sub} of Figure 7 correspond to the improved schedule S^*_{sub} for the reduced problem. Then, the RK-vector $\boldsymbol{x}^*$, leading to the improved schedule S^*, is depicted in Table 4. Applying "Serial_SGS_right($\boldsymbol{x}^*$)", results in the schedule S^* of Figure 8.

Insert Figure 8 About Here

3.5 The decomposition strategy

In our procedure, we iteratively pass through the three steps of the DBH. After each iteration we use the improved schedule S^* to restart the first step. For all iterations we have to predetermine the time interval [pt_1, pt_2] and a stop condition for the genetic algorithm. The settings of these parameters determine the decomposition strategy and will be described in the computational results section.

4 COMPUTATIONAL RESULTS

We have coded the procedure in Visual C++ 6.0 and performed computational tests on an Acer Travelmate 634LC with a Pentium IV 1.8 GHz processor using two datasets. The first one is the well-known PSPLIB dataset (Kolisch and Sprecher 1997) which we use to compare our procedure with other existing procedures from the literature. This dataset contains the subdatasets J30, J60, J90 and J120 that contain problem-instances of 30, 60, 90 and 120 activities respectively. The subdatasets J30, J60 and J90 contain 480 problem-instances, while J120 consists of 600 problem-instances. We also constructed a second dataset RG300 containing 480 large problem instances 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.

4.1 Parameter settings

To test our procedure, we predefine the settings of the parameters of the genetic algorithm. The number of children *nr_children* is fixed at 2, the number of priority values *nr_div* in the diversification step is set equal to (**#Core**)/10 and the threshold for applying diversification is set equal to 2.(**#Core**). We finetune the *popsize* parameter, as its optimal value reflects the extra need for diversification and is related to the stop condition and the size of the problem-instances.

4.2 Comparative computational results of the genetic algorithm

In this section we illustrate that, even without the use of the DBH, our genetic algorithm is very effective. To be able to compare procedures for the RCPSP, Hartmann and Kolisch (2000) presented a methodology in which all procedures can be tested on the PSPLIB-datasets by using 1,000 and 5,000 generated schedules as a stop condition. Kolisch and Hartmann (2004) give an update of these results, and also report results for 50,000 schedules. In tables 5, 6 and 7 we compare our algorithm with these results for the datasets J30, J60 and J120 respectively.

The average deviation from the optimal solution is used as a measure of quality for J30 and the average deviation from the critical path based lower bound for J60 and J120. In each table the heuristics are sorted with respect to increasing deviation for 50,000 schedules. The results for 5,000 schedules are used as a tie-breaker.

Insert Table 5 About Here

The tables reveal that our procedure is capable to report consistently good results. For the J60 and J120 datasets, it outperforms all other procedures. Only for the J30 instances, the procedures of Kochetov and Stolyar (2003) and Debels et al. (2004) report slightly better results for a 50,000 schedules limit. The optimal values of *popsize*, used for the results of tables 5, 6 and 7, are depicted in Table 8. This table reveals that *popsize* is positively related to the schedule limit and negatively related to the number of activities. Thus, the use of a large population avoids, similar to diversification, the creation of a homogeneous population, and this becomes more important for small problem instances and high values for the stop criterion.

Insert Table 6, 7 & 8 About Here

4.3 Computational results of the decomposition-based heuristic

For the experiments with the DBH, we developed a decomposition-strategy as depicted in Figure 9. In the beginning, the GA is used to find a good initial solution for the main problem. Afterwards, we iteratively run the GA on a sub-problem determined by the values for both pt_1 and pt_2. More precisely we run the DBH *nr_sub* times, each time focusing on another part of the main schedule. The X-axis of Figure 9 shows the total makespan of the schedule for the main problem rescaled to the interval [0, 1].

Insert Figure 9 About Here

In the first iteration the main problem is solved by setting the time interval [pt1, pt2] equal to the complete schedule. In the following nr_sub iterations we select [pt1, pt2] such

that the DBH iteratively focuses on a later part of the schedule and also has a small overlap with a part of the sub-schedule of the previous iteration. To get an overlap, we set pt2 - pt1 equal to (2.makespan)/(nr_sub + 1). As an example, if nr_sub = 3 then we construct three reduced problems for which pt2 - pt1 is equal to $2/4$ *makespan*. The procedure focuses iteratively on a later part of the schedule as follows: [pt1, pt2] = [0, $1/2$ *makespan*] for the first sub-problem, [pt1, pt2] = [$1/4$ *makespan*, $3/4$ *makespan*] for the second sub-problem and [pt1, pt2] = [$1/2$ *makespan*, makespan] for the third sub-problem.

To illustrate that the DBH can level up the performance of the GA, it is impossible to use the number of generated schedules as a stop condition since considering a schedule for a sub-problem as a complete schedule would remove the advantage of decomposition. Indeed, decomposition benefits from the fact that the CPU-time needed to build a schedule is proportional to the size of the problem or the value of pt2 - pt1. Therefore, we use the CPU-time as a stop criterion. In our experiments, we use an equal value of popsize for all iterations of the DBH. Since an optimal value for popsize depends on the number of generated schedules, we allocate a time-limit proportional to the value of pt2 - pt1. In doing so, we generate a more or less equal number of schedules in each iteration. In Figure 9 time-allocations are depicted between brackets.

For our experiments, we imposed a stop condition of 0.1 sec, 1 sec and 10 sec. We tested on the PSPLIB-datasets and on the RG300 dataset and the results can be found in Table 8. Both the results for the genetic algorithm (i.e. when nr_sub is fixed at 0) and for the DBH (i.e. when nr_sub $\geq$ 0) are given. The line "Avg.Dev.Ub" shows the average deviation from the best solutions found so far. For the PSPLIB-datasets, these solutions can be found on http://www.bwl.uni-kiel.de/bwlinstitute/Prod/psplib/datasm.html. For RG300 we used the solutions corresponding to a time-limit of 10 sec and optimal values of popsize and nr_sub. The line "<Ub" reports for J60, J90 and J120 the number of problem instances for which the obtained solution is better than the best solution found so far. The lines "popsize" and "nr_sub" report the optimal values for both parameters.

Insert Table 9 About Here

The test results of Table 9 can be interpreted as follows. First, the table reveals that the introduction of the DBH has a beneficial effect for large problem instances. The percentage improvement ("Avg.Dev.UB") for the J30 and J60 instances is only small compared to the

results for the J120 and RG300 instances. The number of improved schedules found, i.e. <UB, increases for large time limits and with *nr_sub* > 0. Second, the optimal size of the population, *popsize*, is similar to the results found in Table 8. The values, however, are not equal, since both tables use another stop criterion. In Table 8, the results were truncated after a pre-defined number of schedules. Table 9 uses a time-limit as a stop criterion. Last, the table reveals that *nr_sub* is negatively related to the time limit, i.e. the more schedules generated, the less beneficial the DBH. Only the J60 instances show some counterintuitive results.

5 CONCLUSIONS

In this paper we developed a new decomposition-based meta-heuristic for the well-known resource-constrained project scheduling problem (RCPSP). We developed a competitive genetic algorithm (GA) and embedded this meta-heuristic in our so-called Decomposition-Based Heuristic (DBH). This heuristic solves sub-problems by iteratively focusing on a different subpart of the schedule. We performed detailed computational results on two problem sets; the well known PSPLIB dataset and a self-made RG300 dataset. Experiments on the PSPLIB dataset revealed that, in general, the GA outperforms all other state-of-the-art procedures (tables 5, 6 and 7). The solution-quality can be further improved when the GA is incorporated in the DBH. Especially for datasets that contain large problem-instances (such as J120 and RG300), it is beneficial to focus on a subpart of the problem. The results in Table 8 also illustrated that for a stop condition of 10 seconds, the DBH generates high-quality solutions that were never found by any other procedure.

Our future research will focus on the decomposition of scheduling problems in two ways. First, we believe that decomposing a problem into smaller sub-problems can be beneficial to other problem types than the RCPSP. Secondly, the decomposition of problems can be used to find exact solutions for the RCPSP, or to incorporate these exact procedures into the DBH approach to find near-optimal solutions in an efficient way.

REFERENCES

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

Baar, T., Brucker, P., Knust, S., 1998. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem, Meta-heuristics: Advances and trends in local search paradigms for optimization, 1–8.

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

Brucker, P., 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.

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

Coelho, J., Tavares, L., 2003. Comparative analysis of meta–heuricstics for the resource constrained project scheduling problem, Technical report, Department of Civil Engineering, Instituto Superior Tecnico, Portugal.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2004. A scatter-search meta-heuristic for the resource-constrained project scheduling problem, European Journal of Operational Research, forthcoming.

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

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

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

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

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

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

Herroelen, W., Demeulemeester, E., De Reyck, B., 1999. A classification scheme for project scheduling. In: Weglarz, J. (Ed.), Project Scheduling – Recent Models, Algorithms and Applications, International Series in Operations Research and Management Science, Kluwer Academic Publishers, Boston, 14, pp. 77-106.

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.

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.

Kochetov, Y., Stolyar, A., 2003. Evolutionary local search with variable neighbourhood for the resource constrained project scheduling problem, Proceedings of the 3rd International Workshop of Computer Science and Information Technologies.

Kolisch, R., 1995. Project scheduling under resource constraints — Efficient heuristics for several problem classes, Physica- Verlag, Heidelberg.

Kolisch, R., 1996a. Efficient priority rules for the resource-constrained project scheduling problem, Journal of Operations Management, 14, 179–192.

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

Kolisch, R., Drexl, A., 1996. Adaptive search for solving hard project scheduling problems, Naval Research Logistics, 43, 23–40.

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

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

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

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

Leon V. J., Ramamoorthy, B., 1995. Strength and adaptability of problem-space based neighbourhoods for resource-constrained scheduling, OR Spektrum, 17, 173–182.

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

Merkle, D., Middendorf, M., Schmeck, H., 2002. Ant colony optimization for resource constrained project scheduling, IEEE Transaction on Evolutionary Computation, 6(4), 333-346.

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

Nonobe, K., Ibaraki, T., 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). In: Ribeiro, C.C., Hansen, P. (Eds.), Essays and Surveys in Meta-heuristics, Kluwer Academic Publishers, Boston, pp. 557-588.

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

Schirmer, A., 2000. Case–based reasoning and improved adaptive search for project scheduling, Naval Research Logistics, 47, 201–222.

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

Tormos, P., Lova, A., 2001. A competitive heuristic solution technique for resource-constrained project scheduling, Annals of Operations Research, 102, 65-81.

Tormos, P., Lova, A., 2003a. An efficient multi-pass heuristic for project scheduling with constrained resources, International Journal of Production Research, 41, 1071-1086.

Tormos, P., Lova, A., 2003b. Integrating heuristics for resource constrained project scheduling: One step forward, Technical report, Department of Statistics and Operations Research, Universidad Polit´ecnica de Valencia.

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

Valls, V., Ballestin, F., Quintanilla, S., 2004b. Justification and RCPSP: A technique that pays, European Journal of Operational Research, Forthcoming.

FIGURE 1

The example project *P*

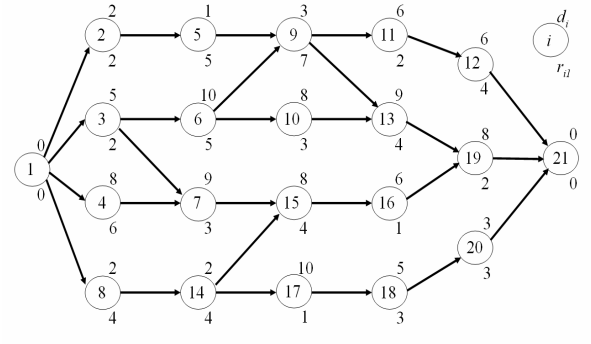

TABLE 1

The RK-vector *x*

Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Priority	0	2	4	5	13	9	8	6	25	16	33	39	38	10	17	31	22	30	42	36	44

.

FIGURE 2

The left-justified schedule and the right-justified schedule

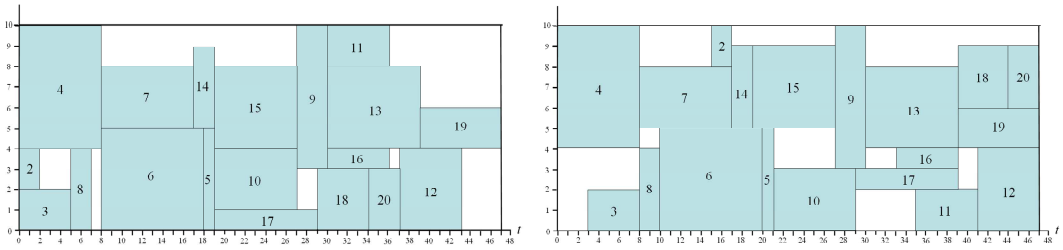

FIGURE 3

The schedule S_f and the corresponding Resource Utilization Profile

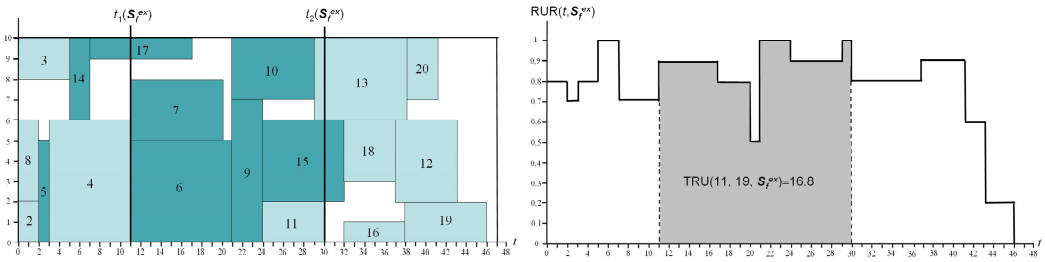

FIGURE 4

The schedule S_m

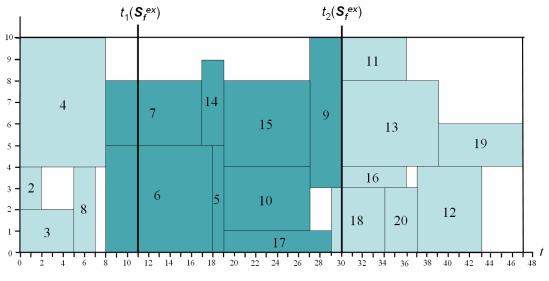

TABLE 2

Calculations of the crossover operator

Act.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
x_f	0	2	5	11	3	21	20	2	24	29	30	43	38	7	32	38	17	37	46	41	46
x_m	0	2	5	8	19	18	17	7	30	27	36	43	39	19	27	36	29	34	47	37	47
x_c	-50	-48	-45	-42	3	21	20	-43	24	29	86	93	89	7	32	86	17	84	97	87	97
$s(S_c)$	0	4	1	0	8	9	10	6	19	19	32	38	27	8	22	30	26	36	36	41	44

FIGURE 5

The schedule S_c

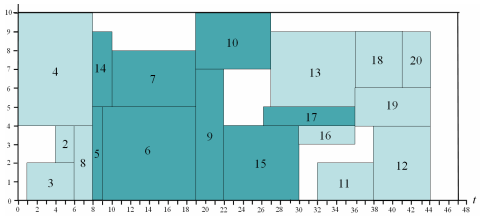

FIGURE 6

The reduced problem P_{sub} and the schedule S_b

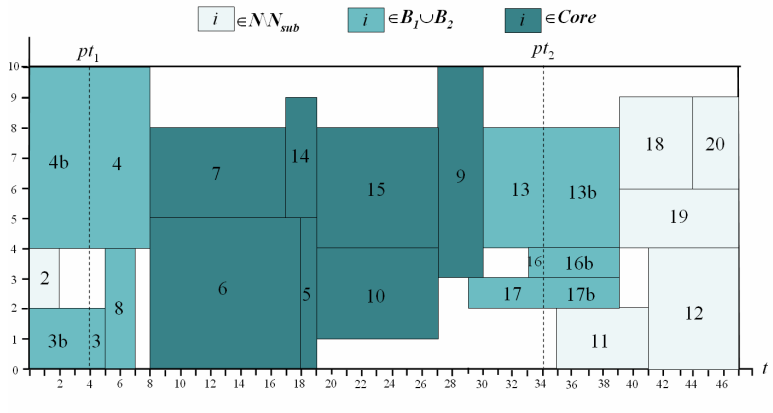

FIGURE 6A

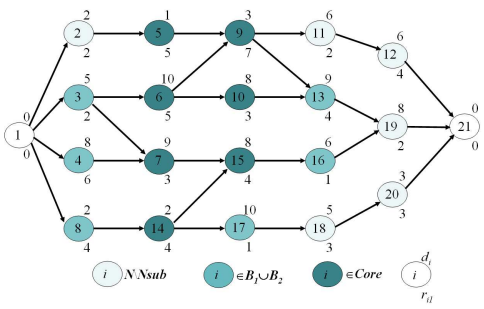

FIGURE 6B

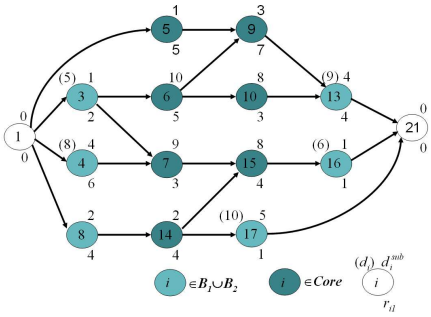

FIGURE 6C

TABLE 3

An RK-representation x_{sub} for the reduced problem P_{sub}

Activity	5	6	7	9	10	14	15
Priority	4	5	5	15	15	3	18

FIGURE 7

The schedule S_{sub} obtained from "Serial_SGS_left(x_{sub})"

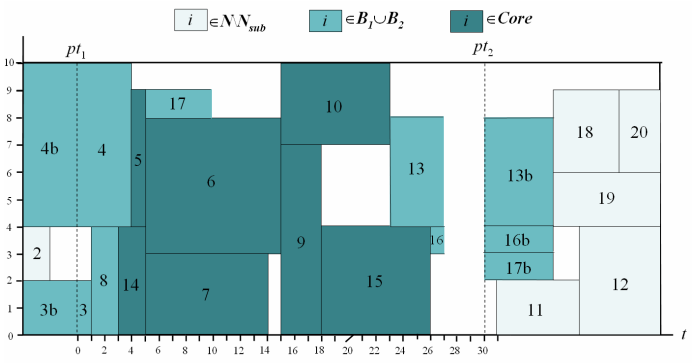

TABLE 4

The resulting RK-representation x^*

Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
Priority	0	2	5	8	9	19	18	7	22	27	41	47	39	9	30	39	39	44	47	47	47

FIGURE 8

The schedule S^* resulting from "Serial_SGS_right(x^*)"

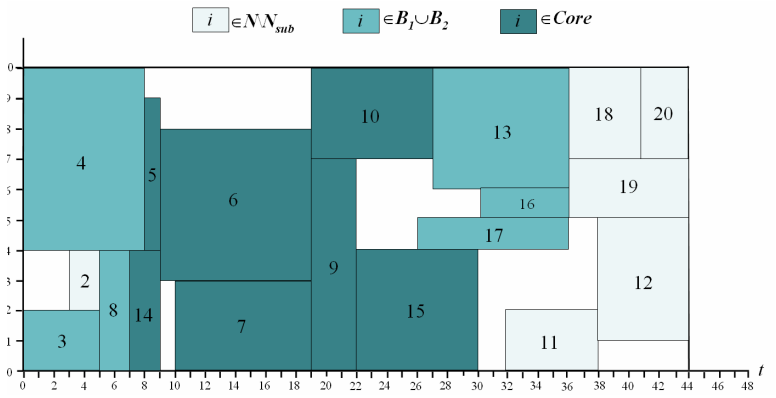

TABLE 5

Average deviations (%) from optimal makespan for J30

	max. #schedules		
Algorithm	1,000	5,000	50,000
Kochetov and Stolyar (2003)	0.10	0.04	0.00
Debels et al. (2004)	0.27	0.11	0.01
Our procedure	**0.15**	**0.04**	**0.02**
Valls, Ballestin and Quintanilla (2002)	0.27	0.06	0.02
Alcaraz and Maroto (2001)	0.33	0.12	-
Valls, Ballestin and Quintanilla.(2004b)	0.34	0.20	0.02
Tormos and Lova (2003b)	0.25	0.13	0.05
Nonobe and Ibaraki (2002)	0.46	0.16	0.05
Tormos and Lova (2001)	0.30	0.16	0.07
Hartmann (2002)	0.38	0.22	0.08
Hartmann (1998)	0.54	0.25	0.08
Tormos and Lova (2003a)	0.30	0.17	0.09
Valls, Ballestin and Quintanilla.(2004b)	0.46	0.28	0.11
Bouleimen and Lecocq (2003)	0.38	0.23	-
Coelho and Tavares (2003)	0.74	0.33	0.16
Schirmer (2000)	0.65	0.44	-
Baar, Brucker and Knust (1998)	0.86	0.44	-
Kolisch and Drexl (1996)	0.74	0.52	-
Hartmann (1998)	1.03	0.56	0.23
Kolisch (1996)	0.83	0.53	0.27
Coelho and Tavares (2003)	0.81	0.54	0.28
Kolisch (1995)	1.44	1.00	0.51
Kolisch (1996b)	1.05	0.78	0.56
Hartmann (1998)	1.38	1.12	0.88
Kolisch (1996a, 1996b)	1.40	1.28	-
Kolisch (1996b)	1.40	1.29	1.13
Kolisch (1995)	1.77	1.48	1.22
Leon and Ramamoorthy (1995)	2.08	1.59	-

TABLE 6

Average deviations (%) from the critical path based lower bound for J60

	max. #schedules		
Algorithm	1,000	5,000	50,000
Our procedure	**11.45**	**10.95**	**10.68**
Debels et al. (2004)	11.73	11.10	10.71
Valls, Ballestin and Quintanilla (2002)	11.56	11.10	10.73
Kochetov and Stolyar (2003)	11.71	11.17	10.74
Valls, Ballestin and Quintanilla.(2004b)	12.21	11.27	10.74
Hartmann (2002)	12.21	11.70	11.21
Hartmann (1998)	12.68	11.89	11.23
Tormos and Lova (2003b)	11.88	11.62	11.36
Tormos and Lova (2003a)	12.14	11.82	11.47
Alcaraz and Maroto (2001)	12.57	11.86	-
Tormos and Lova (2001)	12.18	11.87	11.54
Bouleimen and Lecocq (2003)	12.75	11.90	-
Nonobe and Ibaraki (2002)	12.97	12.18	11.58
Valls, Ballestin and Quintanilla.(2004b)	12.73	12.35	11.94
Schirmer (2000)	12.94	12.58	-
Coelho and Tavares (2003)	13.28	12.63	11.94
Hartmann (1998)	14.68	13.32	12.25
Hartmann (1998)	13.30	12.74	12.26
Kolisch and Drexl (1996)	13.51	13.06	-
Kolisch (1996a, 1996b)	13.66	13.21	-
Coelho and Tavares (2003)	13.80	13.31	12.83
Kolisch (1996b)	13.75	13.34	12.84
Kolisch (1996b)	13.59	13.23	12.85
Baar, Brucker and Knust (1998)	13.80	13.48	-
Leon and Ramamoorthy (1995)	14.33	13.49	-
Kolisch (1996b)	13.96	13.53	12.97
Kolisch (1995)	14.89	14.30	13.66
Kolisch (1995)	15.94	15.17	14.22

TABLE 7

Average deviations (%) from the critical path based lower bound for J120

	max. #schedules		
Algorithm	1,000	5,000	50,000
Our procedure	**34.19**	**32.34**	**30.82**
Valls, Ballestin and Quintanilla (2002)	34.07	32.54	31.24
Debels et al. (2004)	35.22	33.10	31.57
Valls, Ballestin and Quintanilla.(2004b)	35.39	33.24	31.58
Kochetov and Stolyar (2003)	34.74	33.36	32.06
Valls, Ballestin and Quintanilla.(2004b)	35.18	34.02	32.81
Hartmann (2002)	37.19	35.39	33.21
Tormos and Lova (2003b)	35.01	34.41	33.71
Merkle, Middendorf and Schmeck (2002)	-	35.43	-
Hartmann (1998)	39.37	36.74	34.03
Tormos and Lova (2003a)	36.24	35.56	34.77
Tormos and Lova (2001)	36.49	35.81	35.01
Alcaraz and Maroto (2001)	39.36	36.57	-
Nonobe and Ibaraki (2002)	40.86	37.88	35.85
Coelho and Tavares (2003)	39.97	38.41	36.44
Valls, Ballestin and Quintanilla.(2004b)	38.21	37.47	36.46
Bouleimen and Lecocq (2003)	42.81	37.68	-
Hartmann (1998)	39.93	38.49	36.51
Schirmer (2000)	39.85	38.70	-
Kolisch (1996b)	39.60	38.75	37.74
Kolisch (1996a, 1996b)	39.65	38.77	-
Hartmann (1998)	45.82	42.25	38.83
Kolisch (1996b)	41.27	40.38	39.34
Kolisch and Drexl (1996)	41.37	40.45	-
Coelho and Tavares (2003)	41.36	40.46	39.41
Leon and Ramamoorthy (1995)	42.91	40.69	-
Kolisch (1996b)	42.84	41.84	40.63
Kolisch (1995)	44.46	43.05	41.44
Kolisch (1995)	49.25	47.61	45.60

TABLE 8

Optimal values of *popsize*

	max. #schedules		
Dataset	1,000	5,000	50,000
J30	45	100	480
J60	30	90	480
J120	20	75	360

FIGURE 9

The decomposition-strategy

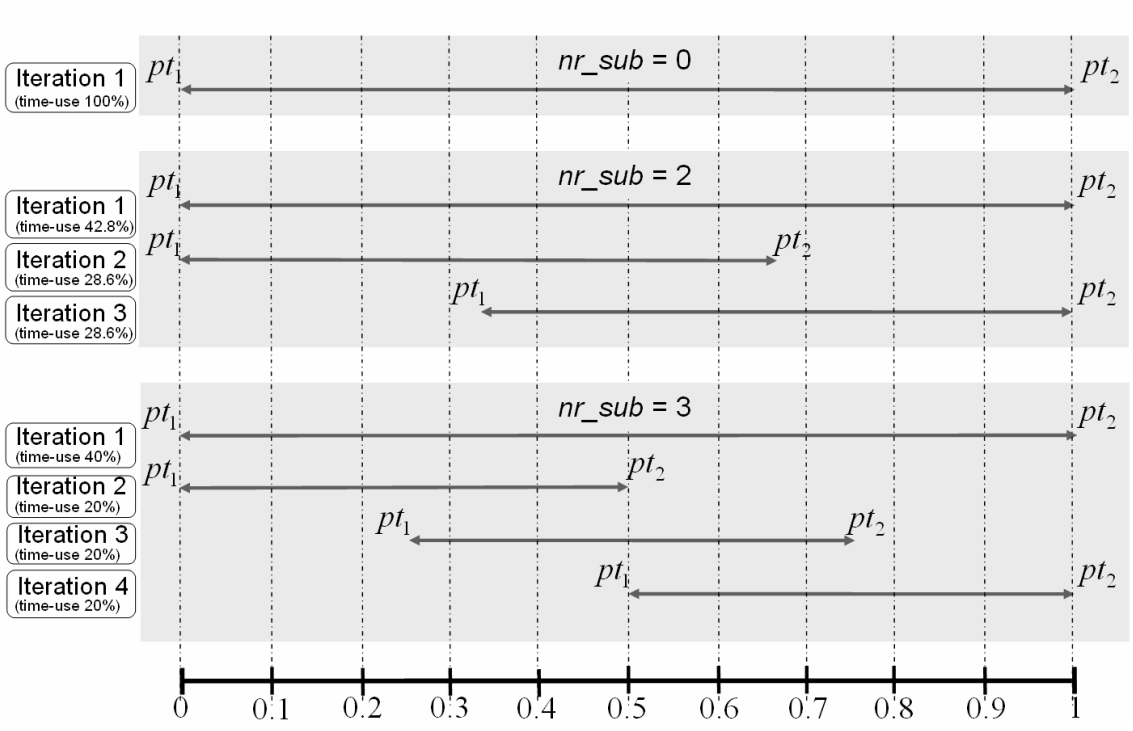

TABLE 9

Overview of the results for the DBH

		nr_sub = 0			*nr_sub* ≥ 0		
		0.1 sec	1 sec	10 sec	0.1 sec	1 sec	10 sec
	Avg.Dev.Ub	0.05%	0.01%	0.00%	0.04%	0.01%	0.00%
J30	*popsize*	240	1,000	3,600	160	1,000	3,600
	nr_sub	0	0	0	2	0	0
	Avg.Dev.Ub	0.37%	0.18%	0.08%	0.37%	0.16%	0.07%
	<Ub	0	1	1	0	0	3
J60	*popsize*	80	480	2,000	80	400	2,200
	nr_sub	0	0	0	0	2	2
	Avg.Dev.Ub	0.62%	0.23%	0.05%	0.60%	0.21%	0.02%
	<Ub	0	0	11	1	4	20
J90	*popsize*	45	240	2,000	30	220	1,800
	nr_sub	0	0	0	4	2	2
	Avg.Dev.Ub	2.15%	0.82%	0.15%	2.04%	0.78%	0.10%
	<Ub	0	5	63	0	2	67
J120	*popsize*	25	180	1,400	25	120	1,400
	nr_sub	0	0	0	4	3	2
	Avg.Dev.Ub	2.76%	1.37%	0.27%	2.10%	0.98%	0.00%
RG300	*popsize*	10	30	240	10	30	240
	nr_sub	0	0	0	12	8	5