Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem

Dieter Debels, Mario Vanhoucke,

Operations Research
Vol. 55, No. 3, May–June 2007, pp. 457–469
ISSN 0030-364X | EISSN 1526-5463 | 07 | 5503 | 0457

DOI 10.1287/opre.1060.0358

A Decomposition-Based Genetic Algorithm for the Resource-Constrained Project-Scheduling Problem

Dieter Debels
Faculty of Economics and Business Administration, Ghent University, Ghent, Belgium, dieter.debels@ugent.be

Mario Vanhoucke
Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Ghent, Belgium, mario.vanhoucke@ugent.be

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 that is able to provide near-optimal heuristic solutions. This GA procedure has been extended by a so-called decomposition-based genetic algorithm (DBGA) that iteratively solves subparts of the project. We present computational experiments on two data sets. The first benchmark set is used to illustrate the performance of both the GA and the DBGA. 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 the GA outperforms all state-of-the-art heuristics and that the DBGA further improves the performance of the GA.

Subject classifications: production/scheduling: approximations/heuristic; project management: resource constraints.
Area of review: Optimization.
History: Received January 2005; revision received January 2006; accepted April 2006.

1. Introduction

We study the resource-constrained project-scheduling problem (RCPSP) denoted as $m, 1|cpm|C_{\max}$, using the classification scheme of Herroelen et al. (1999). The RCPSP can be stated as follows. In a project network in AoN format $G(\mathbf{N}, \mathbf{A})$, we have a set of nodes $\mathbf{N}$ and a set of pairs $\mathbf{A}$, representing the direct precedence relations. The set $\mathbf{N}$ contains n activities, numbered from 1 to n ($\#\mathbf{N} = n$). The set of pairs $\mathbf{A}^+$ adds the transitive precedence relations to $\mathbf{A}$. Furthermore, we have a set of resources $\mathbf{R}$, and for each resource type $k \in \mathbf{R}$ there is a constant availability a_k throughout the project horizon. Each activity $i \in \mathbf{N}$ has a deterministic duration $d_i \in \mathbb{N}$ and requires $r_{ik} \in \mathbb{N}$ units of resource type k. We assume that $r_{ik} \leqslant a_k$ for $i \in \mathbf{N}$ and $k \in \mathbf{R}$. 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 $\mathbf{s}(S) = (s_1, \ldots, s_n)$, which implies an n-vector of finish times $\mathbf{f}(S)$, where $f_i = s_i + d_i \ \forall\, i \in \mathbf{N}$. 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 et al. (1998), Icmeli et al. (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 of resolving 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 metaheuristic procedures of Alcaraz and Maroto (2001), Tormos and Lova (2001, 2003a), Valls et al. (2002, 2004, 2005), Valls et al. (2003), and Debels et al. (2006) 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. (2006) 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 RCPSP. Furthermore, this algorithm will be embedded in a heuristic search procedure (subsequently referred to as the decomposition-based genetic algorithm (DBGA)) that iteratively solves subparts of the problem under study. More precisely, the procedure selects a subproblem from a feasible solution for the main problem under study. The GA is then used to find a high-quality solution for the subproblem that can be reincorporated into the solution of the main problem. The idea of focusing on subparts of the problem is not new and has already been elaborated on by Mausser and Lawrence (1997), Palpant et al. (2004), and Sprecher (2002). These authors all present a methodology in which they use subparts of an initial schedule to create subproblems, which can be solved by an exact or heuristic solution procedure to improve the current schedule. However, we believe that our DBGA outperforms these decomposition-based procedures thanks to the well-thought-out construction of the subproblems, which will be described later. Mausser and Lawrence (1997) only consider so-called block structures as possible subproblems, which makes their decomposition method rather inflexible because they have less control of the size of the subproblems. We believe that the decomposition method of Sprecher (2002) offers more flexibility than that of Mausser and Lawrence (1997). However, a major disadvantage is that the resulting main schedule (obtained by solving all subproblems) may be a schedule that is worse than the initial schedule. Palpant et al. (2004) construct a subproblem that can be classified as an RCPSP with time windows and varying resource availability. However, their specific approach does not guarantee that an improvement found in the subproblem also leads to an improvement of the main problem. Moreover, the authors solve this problem with the ILOG SCHEDULER 5.0, and hence, they fail in reporting results that can compete with the currently best existing procedures.

This paper is organized as follows. In §2, we describe our GA approach for the RCPSP in detail. In §3, we explain the different steps of our DBGA. Section 4 discusses extensive computational results of the GA and the DBGA 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 metaheuristic for the RCPSP. The *schedule representation* can be seen as an encoding of a schedule. To decode the schedule representation into a schedule S, identified by $\mathbf{s}(S)$ and $\mathbf{f}(S)$, 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 widespread. 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 $\mathbf{x} \in \mathbb{R}^n$ 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. (2006) recently illustrated that the RK also leads to promising results thanks to the use of the *topological ordering* (TO) notation (Valls et al. 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 because we rely on two types of schedules: left justified and right justified. Hence, to fully exploit the advantages of the iterative forward/backward scheduling technique, our procedure uses right-justified schedules to generate left-justified children and left-justified schedules to generate right-justified children. Therefore, we embed the TO condition in the RK representation as follows:

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

In §2.2, we rely on this specific TO condition to perform the crossover operations in an effective way.

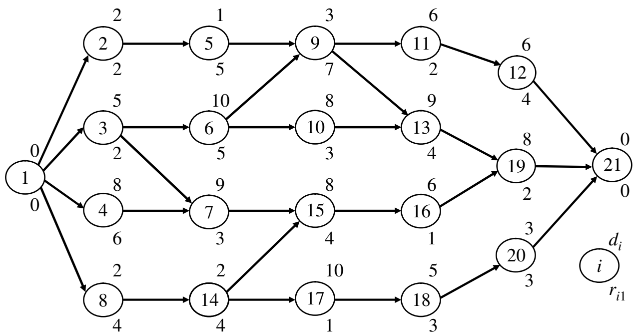
In the literature there also exist different types of SGSs. The serial SGS constructs an active schedule by scheduling each activity one at a time and as soon as possible within the precedence and resource constraints. Alternatively, a parallel SGS could be used. However, Kolisch (1996b) has shown that, contrary 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($\mathbf{x}$)" and "Serial_SGS_right($\mathbf{x}$)," 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($\mathbf{x}$), while the opposite will be true for Serial_SGS_right($\mathbf{x}$).

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 $\mathbf{x}$. 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($\mathbf{x}$) leads to the left-justified schedule of Figure 2, while calling Serial_SGS_right($\mathbf{x}$) brings us to the right-justified schedule of Figure 3.

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 a *genetic algorithm* (GA). In a

Figure 1. The example project **P**.



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 GA, 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. Therefore, the left (right) population elements are transformed in right-justified (left-justified) schedules by means of the procedure Serial_SGS_right($\mathbf{x}$) (Serial_SGS_left($\mathbf{x}$)). In doing so, we can fully exploit the advantages of the iterative forward/backward scheduling technique in our metaheuristic framework. 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( )
Step 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 $\mathbf{x}_c$.
1.1.3. Diversification of $\mathbf{x}_c$.
1.1.4. $Child_{a,b}$:= Serial_SGS_right($\mathbf{x}_c$).
1.2. Select the best child $Child_a^{best}$.
1.3. Replace $PopR_a$ by $Child_a^{best}$.

Procedure Update_PopL( )
Step 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 $\mathbf{x}_c$.
1.1.3. Diversification of $\mathbf{x}_c$.
1.1.4. $Child_{a,b}$:= Serial_SGS_left($\mathbf{x}_c$).
1.2. Select the best child $Child_a^{best}$.
1.3 Replace $PopL_a$ by $Child_a^{best}$.

Table 1. The RK vector **x**.

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

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

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 ath element of *PopR*). The second parent is selected using the two-tournament selection where two population

Figure 2. Left-justified schedule.

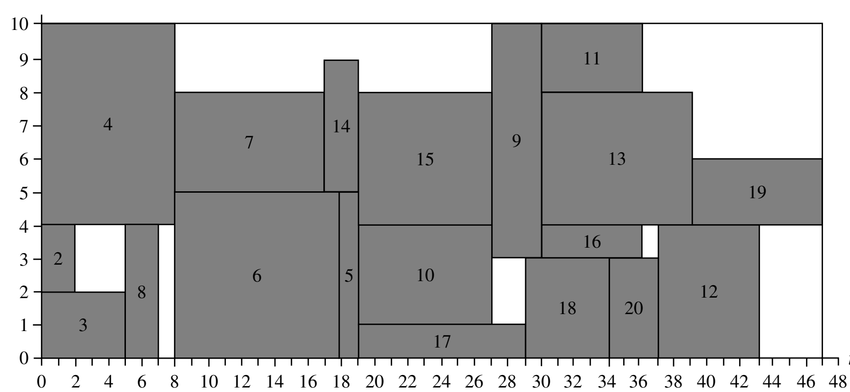
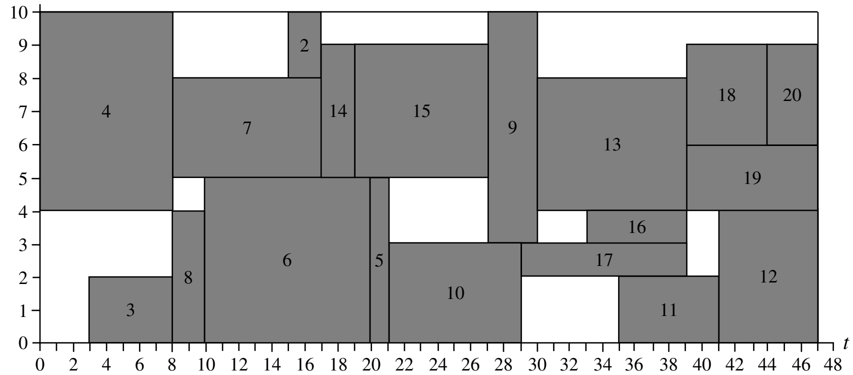


Figure 3. Right-justified schedule.



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 $\mathbf{x}_f$ of the father S_f and $\mathbf{x}_m$ of the mother S_m in a new RK vector $\mathbf{x}_c$ 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 et al. (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

$$\mathrm{RUR}(t, S) = \frac{1}{K} \sum_{j \in active(t, S)} \sum_{k=1}^{K} \frac{r_{jk}}{a_k}. \tag{1}$$

The RUR allows us to select time intervals for which the resource utilization is high, so-called peaks (Valls et al. 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 $(1/4) \cdot \text{makespan}(S)$ and $(3/4) \cdot \text{makespan}(S)$ and calculate the *total resource utilization* (TRU) of a subschedule of S with start time t and length l:

$$\mathrm{TRU}(t, l, S) = \sum_{\text{time}=t}^{t+l-1} \mathrm{RUR}(\text{time}, S). \tag{2}$$

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 $\mathrm{TRU}(t, l, S)$ 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 $t_1(S_f) \leqslant x_{mi} \leqslant t_2(S_f) \Rightarrow x_{ci} := x_{fi}$.

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

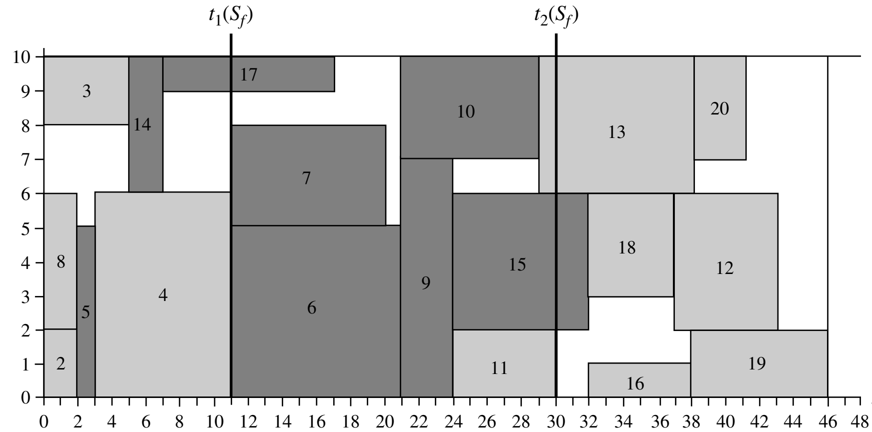
The use of the TO condition allows the crossover operator to replace the (weak) subschedules with a low resource utilization of the father S_f by better-corresponding subschedules of the mother S_m. To prevent the relative priority structure between the activities of a case from being mixed with priority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

Consider Figure 4, which represents an example of a left-justified schedule S_f of *PopL*. To calculate the crossover points $t_1(S_f)$ and $t_2(S_f)$ of a peak, we first compose the *resource utilization profile* of Figure 5 by calculating the RUR at each time instant t. If l has been chosen randomly to 19, then $\mathrm{TRU}(t, 19, S_f)$ has a maximal value of 16.8 when $t = 11$. Consequently, $t_1(S_f)$ equals 11 and $t_2(S_f)$ equals 30.

We assume a mother schedule $S_m \in PopL$, as depicted in Figure 6. Table 2 displays the RK vectors $\mathbf{x}_f = \mathbf{f}(S_f)$ and $\mathbf{x}_m = \mathbf{f}(S_m)$. Because 11 and 30 are the crossover points of the father, the RK vector $\mathbf{x}_c$ of the child is calculated as demonstrated in Table 2. The dark-coloured activities 5, 6, 7, 9, 10, 14, 15, and 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 $\mathbf{x}_c$. However, a large constant c needs to be subtracted for activities of Case 1, and added for activities of Case 3, to avoid mixing the priority structures of Case 1, Case 2, and Case 3. In our example, we set c at 50.

The corresponding child schedule can be constructed based on the calculated RK vector $\mathbf{x}_c$ of Table 2. Because the parents belong to *PopL*, we call Serial_SGS_right($\mathbf{x}_c$) to build the right-justified schedule S_c of Figure 7. As this schedule is a candidate to enter *PopR*, the RK vector is set equal to $\mathbf{s}(S_c)$.

Figure 4. The schedule S_f of the father.



Diversification. We implemented a diversification methodology 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 because it only operates on a child when it originates from two parents who are not mutually diverse. To define whether the parents are sufficiently diverse, we need a threshold τ and a distance measure. Because we employ $\mathbf{s}(S)$ or $\mathbf{f}(S)$, 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

$$\tau > \frac{1}{n}\sum_{i=1}^{n}|x_{fi} - x_{mi}|$$

and is exerted on the child $\mathbf{x}_c$ 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. To make the GA successful, the *survival-of-the-fittest* principle should be embedded. Good children should have a better chance to enter the new generation than inferior ones 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 $(Child_{a,1}, \ldots, Child_{a,nr\_children})$. 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, to avoid loosening 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.

Figure 5. The resource utilization profile of schedule S_f.

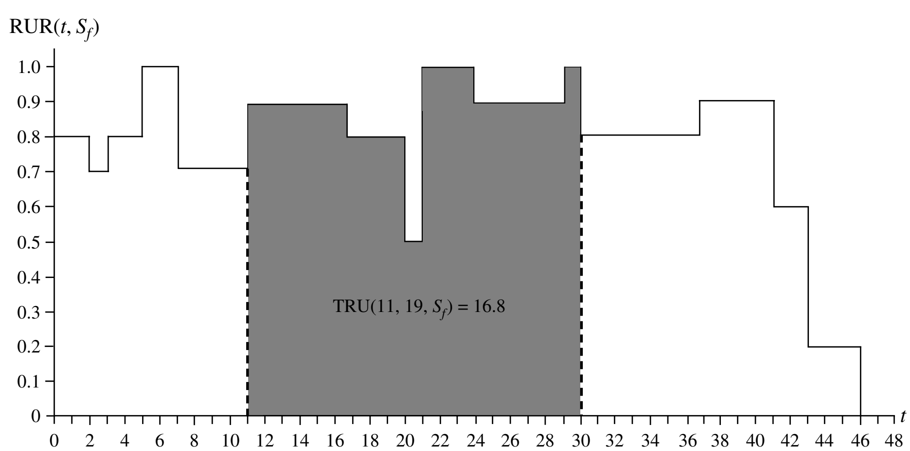
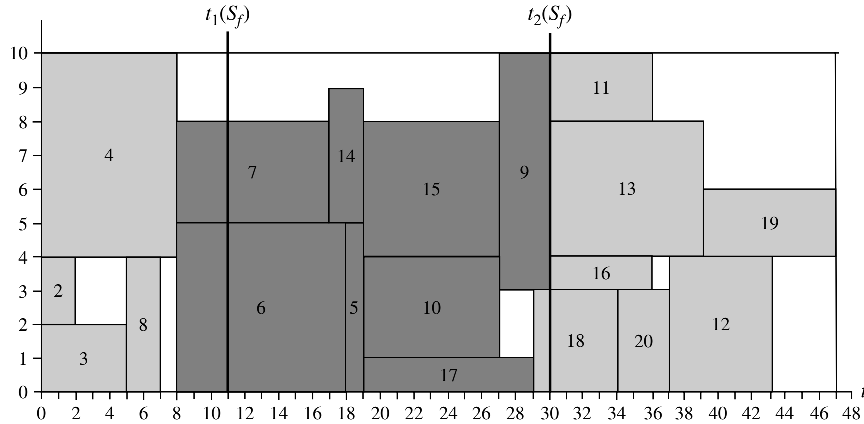


Figure 6. The schedule S_m of the mother.



3. The Decomposition-Based Genetic Algorithm

3.1. Introduction

The GA of the previous section can be used as a subroutine in a heuristic decomposition approach. This decomposition-based genetic algorithm (DBGA) runs as follows: An intermediate solution of the main problem (the main schedule) is used as a start base to derive a smaller subproblem. This subproblem is then solved with the GA, resulting in an improved subschedule. Afterwards, this improved subschedule 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 the Subproblem*. This subroutine transforms a schedule S of the main problem $\mathbf{P}$ into an intermediate schedule S_b and uses that schedule to construct a smaller subproblem $\mathbf{P}_{sub}$ and an initial subschedule S_{sub}^{in}.

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

Step 3. *Merge*. This subroutine embeds the improved subschedule 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 DBGA in detail.

3.2. Construct the Subproblem

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 subproblem $\mathbf{P}_{sub}$. The details are summarized in the pseudo-code below, using the following symbols. Problem $\mathbf{P} = G(\mathbf{N}, \mathbf{A})$ with start schedule S will, based on the transformed intermediate schedule S_b, be reduced to subproblem $\mathbf{P}_{sub} = G(\mathbf{N}_{sub}, \mathbf{A}_{sub})$. The set of nodes $\mathbf{N}_{sub}$ of subproblem $\mathbf{P}_{sub}$ is further subdivided in the sets $\mathbf{B}_1$, $\mathbf{B}_2$, and ***Core***. These sets are used in the GA approach as will be described in §3.3.

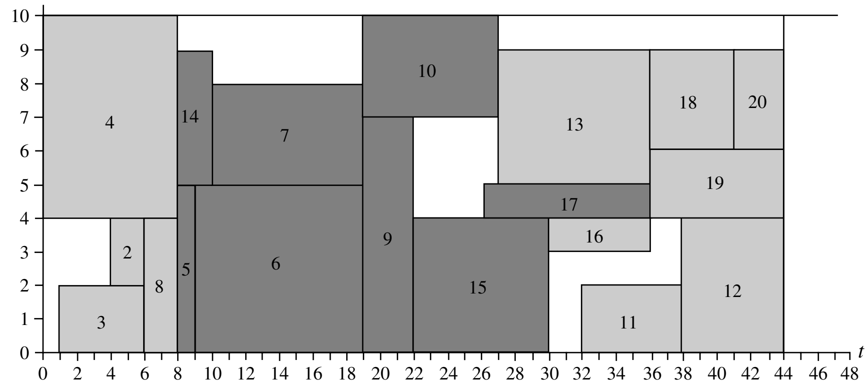
Procedure Build_subproblem($S, pt_1, pt_2, \mathbf{P}$)
Step 1. $\mathbf{B}_1 := \mathbf{B}_2 := \boldsymbol{Core} := \varnothing$.
Step 2. Transform S into S_b.
Step 3. Determine $lft = \max\{f_i \mid s_i < pt_1\}$ and $est = \min\{s_i \mid f_i > pt_2\}$.
Step 4. Construct the subproblem $\mathbf{P}_{sub} = G(\mathbf{N}_{sub}, \mathbf{A}_{sub})$.
Step 5. For $[i = 2, n-1]$,
If $(f_i(S_b) > pt_1$ and $s_i(S_b) < pt_2)$
If $(s_i(S_b) \geqslant lft$ and $f_i(S_b) \leqslant est)$, then
$\boldsymbol{Core} := \boldsymbol{Core} \cup \{i\}$
If $(s_i(S_b) < lft)$, then $\mathbf{B}_1 := \mathbf{B}_1 \cup \{i\}$
If $(f_i(S_b) > est)$, then $\mathbf{B}_2 := \mathbf{B}_2 \cup \{i\}$.

Step 1 initializes the sets of activities $\mathbf{B}_1$, $\mathbf{B}_2$, 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 doing so, the number of active activities scheduled within the time interval $[pt_1, pt_2]$ is reduced compared to the schedule S, which makes it more likely to find improvements in that part of the schedule. In Step 3, we calculate time window $[lft, est]$ with $lft \geqslant pt_1$ and $est \leqslant pt_2$. More precisely, lft denotes the latest finish time of all

Table 2. Calculations of the crossover operator.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$\mathbf{x}_f$	0	2	5	11	3	21	20	2	24	29	30	43	38	7	32	38	17	37	46	41	46
$\mathbf{x}_m$	0	2	5	8	19	18	17	7	30	27	36	43	39	19	27	36	29	34	47	37	47
$\mathbf{x}_c$	−50	−48	−45	−42	3	21	20	−43	24	29	86	93	89	7	32	86	17	84	97	87	97
$\mathbf{s}(S_c)$	0	4	1	0	8	9	10	6	19	19	32	38	27	8	22	30	26	36	36	41	44

Figure 7. The schedule S_c of the child.



activities i in schedule S_b for which $s_i \leqslant pt_1$, while *est* denotes the earliest start time of all activities i in schedule S_b for which $f_i \geqslant pt_2$.

In Step 4, we construct the subproblem $\mathbf{P}_{sub} = G(\mathbf{N}_{sub}, \mathbf{A}_{sub})$, defined by the set of activities $\mathbf{N}_{sub}$ and the set of activities $\mathbf{A}_{sub}$. $\mathbf{N}_{sub}$ contains the dummy activities zero 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 $\mathbf{A}^+_{sub}$ equals $\{(i,j) \in \mathbf{A}^+ \mid i, j \in \mathbf{N}_{sub}\}$. The removal of the transitive arcs from $\mathbf{A}^+_{sub}$ results in $\mathbf{A}_{sub}$. The durations d_i^{sub} for all activities $i \in \mathbf{N}_{sub}$ 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, except when i is active at pt_1 or pt_2. In Step 5, the set $\mathbf{N}_{sub}$ is further subdivided in the sets ***Core***, $\mathbf{B}_1$, and $\mathbf{B}_2$. ***Core*** contains the activities that are completely active within [*lft*, *est*]. The remaining activities belong to either $\mathbf{B}_1$ or $\mathbf{B}_2$, depending on whether they are scheduled before *lft* or after *est*.

In our example, we use the right-justified schedule of Figure 3 as S. The time interval $[pt_1, pt_2]$ is set randomly at $[4, 34]$. Figure 8 shows the steps to obtain the resulting subproblem $\mathbf{P}_{sub}$. The intermediate schedule S_b, displayed in Figure 8(a), 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. Twenty-nine 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 $\mathbf{N}_{sub}$, 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*** because 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 $\mathbf{B}_1$ while the activities 13, 16, and 17, that start after time instant 29, belong to $\mathbf{B}_2$. Figure 8(b) displays the network of Figure 1 and the different sets $\mathbf{B}_1$, $\mathbf{B}_2$, ***Core***, and $\mathbf{N}_{sub}$. Figure 8(c) omits the activities not belonging to $\mathbf{N}_{sub}$, and represents the problem $\mathbf{P}_{sub}$. 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 Subproblem

After the construction of a subproblem $\mathbf{P}_{sub}$, this subproblem is the subject of the GA described in §2.2 to find an improved subschedule S^*_{sub}. However, some modifications to the schedule generation schemes (i.e., subroutines Serial_SGS_left($\mathbf{x}$) and Serial_SGS_right($\mathbf{x}$) of §2.2) are necessary to make the GA applicable for the DBGA. More precisely, the activities of $\mathbf{B}_1$, $\mathbf{B}_2$, and ***Core*** need to be treated differently. The serial schedule generation scheme to construct a left-justified subschedule only schedules the activities of $\mathbf{N}_{sub}$, based on the random key vector $\mathbf{x}_{sub}$, and is applied in four consecutive steps:

Step 1. Schedule the activities of $\mathbf{B}_1$ 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 $\mathbf{B}_2$ in a sequence determined by the start times of S_b and by using the original durations d_i.

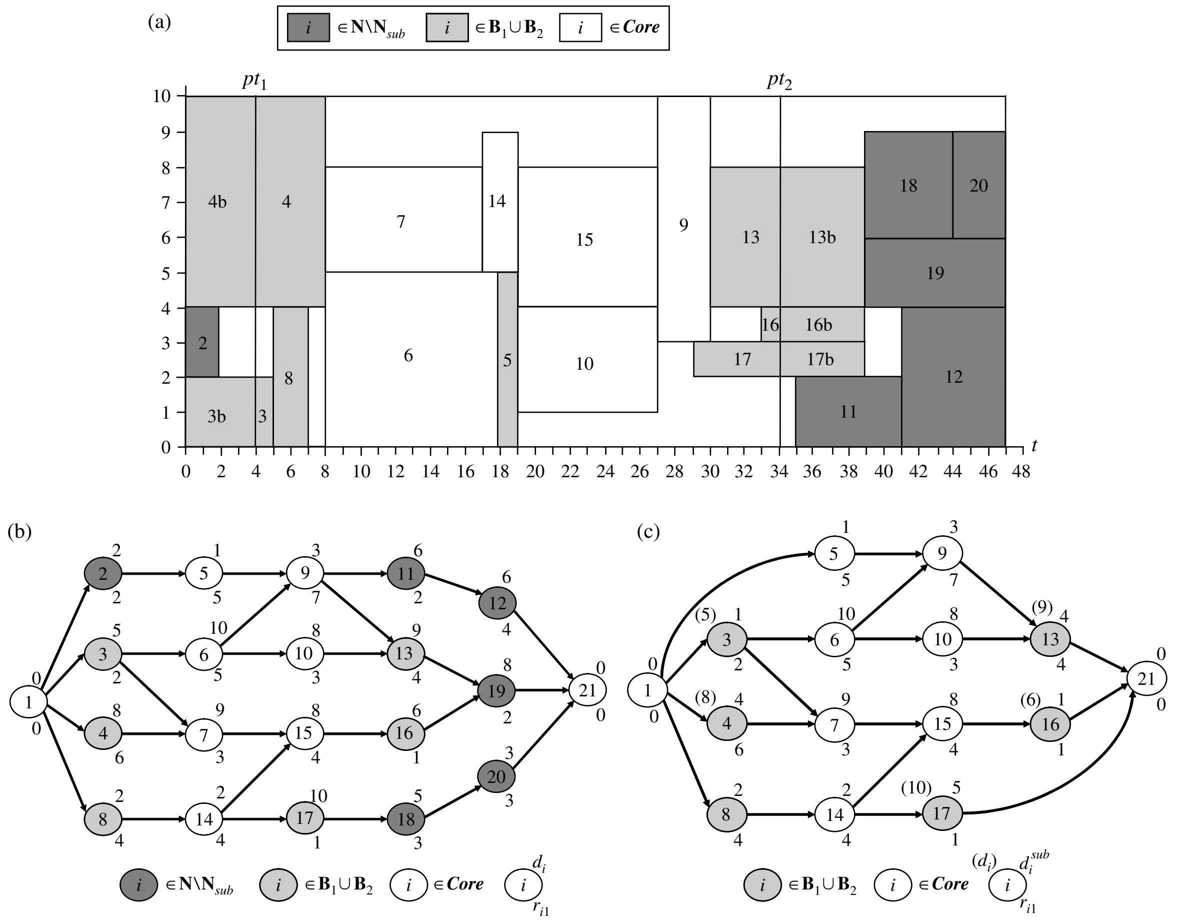
Step 4. Replace the durations of $\mathbf{B}_2$ by d_i^{sub}.

The reason to use the original durations d_i to schedule the activities of $\mathbf{B}_2$ is to obtain a schedule that can be incorporated easily into the schedule S_b without incurring preemption as will be described in the merge step of §3.4. After Step 4, we obtain a feasible schedule for the subproblem.

The construction of a right-justified schedule works oppositely and schedules the activities of $\mathbf{B}_2$, ***Core***, and $\mathbf{B}_1$ backwards in four consecutive steps. The sequence to schedule $\mathbf{B}_1$ and $\mathbf{B}_2$ is determined by the finish times of S_b.

Figure 9 displays a left-justified schedule S_{sub} of $\mathbf{P}_{sub}$ based on the RK vector $\mathbf{x}_{sub}$ of Table 3. First, activities 3, 4, and 8 of set $\mathbf{B}_1$ are scheduled in a sequence determined by the start times of S_b. Then, the dark-coloured activities of

Figure 8. The reduced problem $\mathbf{P}_{sub}$ and the schedule S_b.



Core are scheduled in a sequence based on the RK vector $\mathbf{x}_{sub}$. 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 three time units compared to the initial schedule embedded in S_b within the time interval $[4, 34]$.

3.4. Merge

This subroutine embeds the improved subschedule S^*_{sub} into the intermediate schedule S_b, leading to a new schedule S^*

Figure 9. The schedule S_{sub} obtained from Serial_SGS_left($\mathbf{x}_{sub}$).

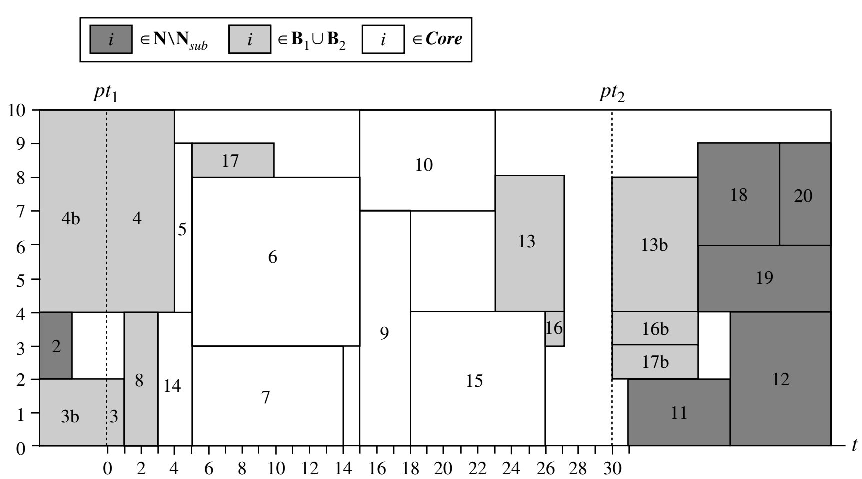


Table 3. An RK representation $\mathbf{x}_{sub}$ for the reduced problem $\mathbf{P}_{sub}$.

i	5	6	7	9	10	14	15
$\mathbf{x}_{sub}$	4	5	5	15	15	3	18

with an overall improvement of the total makespan. To do so, the merge subroutine creates an RK vector $\mathbf{x}^\star$ as follows:

1. For $i \in \boldsymbol{Core}$, $x_i^* = pt_1 + f_i(S_{sub}^*)$.
2. For $i \in \mathbf{N}\backslash\boldsymbol{Core}$, $x_i^* = f_i(S_b)$.

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

Let the schedule S_{sub} of Figure 9 correspond to the improved schedule S_{sub}^* for the reduced problem. Then, the RK vector $\mathbf{x}^\star$, leading to the improved schedule S^*, is depicted in Table 4. Applying Serial_SGS_right($\mathbf{x}^\star$), results in the schedule S^* of Figure 10.

3.5. The Decomposition Strategy

In our procedure, we iteratively pass through the three steps of the DBGA. 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 GA. The settings of these parameters determine the decomposition strategy and will be described in the next 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 data sets. The first one is the well-known PSPLIB data set (Kolisch and Sprecher 1997), which we use to compare our procedure with other existing procedures from the literature. This data set contains the subdata sets J30, J60, J90, and J120 with problem instances of 30, 60, 90, and 120 activities, respectively.

Table 4. The resulting RK representation $\mathbf{x}^*$.

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$\mathbf{x}^*$	0	2	5	8	9	19	18	7	22	27	41	47	39	9	30	39	39	44	47	47	47

The subdata sets J30, J60, and J90 contain 480 problem instances, while J120 consists of 600 problem instances. We also constructed a second data set RG300 containing 480 large problem instances using RanGen (Demeulemeester et al. 2003). Each instance contains 300 activities and four resources. The order strength is set at 0.25, 0.50, or 0.75; the 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. The RG300 problem instances are available at www.projectmanagement.ugent.be/RG300Instances.php.

4.1. Parameter Settings

To test our procedure, we predefine the settings of the parameters of the GA. 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 τ for applying diversification is set equal to 2.(**#Core**). Note that **#Core** represents the number of activities in the subproblem that belong to ***Core***. We fine-tune 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. Computational Results of the GA and the DBGA

For the experiments with the DBGA, we developed a decomposition strategy, as depicted in Figure 11. 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 subproblem determined by the values for both pt_1 and pt_2.

Figure 10. The schedule S^* resulting from Serial_SGS_right($\mathbf{x}^\star$).

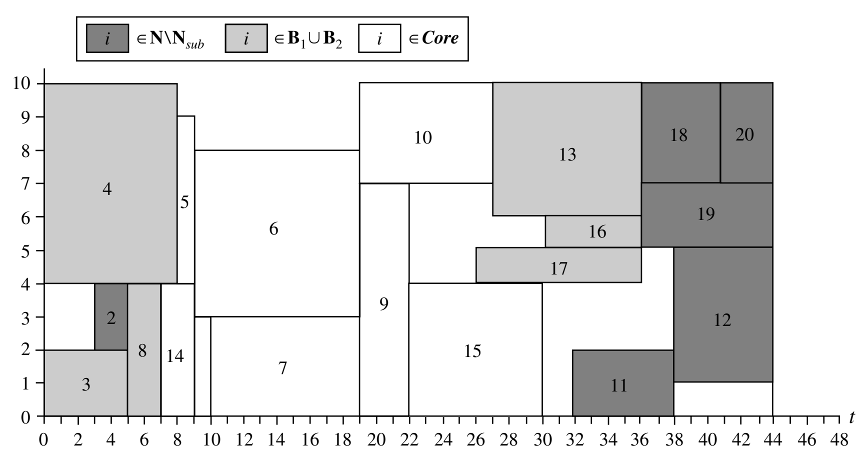
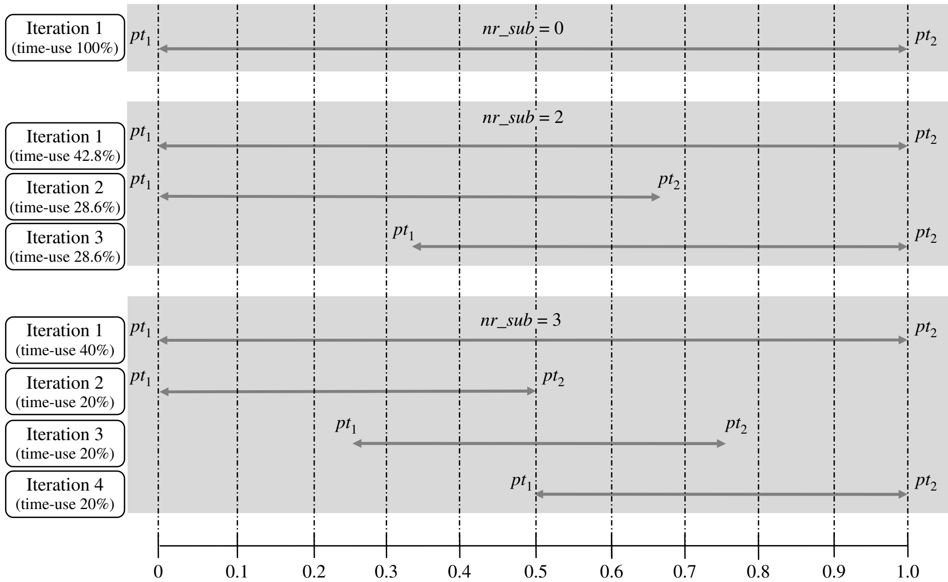


Figure 11. The decomposition strategy.



More precisely, we run the DBGA *nr_sub* times, each time focusing on another part of the main schedule. The X-axis of Figure 11 shows the total makespan of the schedule for the main problem rescaled to the interval [0, 1].

In the first iteration, the main problem is solved by setting the time interval $[pt_1, pt_2]$ equal to the complete schedule. In the following *nr_sub* iterations, we select $[pt_1, pt_2]$ such that the DBGA iteratively focuses on a later part of the schedule and also has a small overlap with a part of the subschedule of the previous iteration. To get an overlap, we set $pt_2 - pt_1$ equal to $(2.makespan)/(nr\_sub + 1)$. As an example, if $nr\_sub = 3$, then we construct three reduced

Table 5. Overview of the results for the GA and the DBGA.

		GA			DBGA		
		1,000	5,000	50,000	1,000	5,000	50,000
J30	Avg.CPU	0.012s	0.055s	0.520s	0.012s	0.055s	0.520s
	Avg.Dev.Ub	0.15%	0.04%	0.02%	0.12%	0.04%	0.02%
	Avg.Dev.Lb	13.60%	13.44%	13.41%	13.55%	13.44%	13.41%
	popsize	45	100	480	42	100	480
	nr_sub	0	0	0	2	0	0
J60	Avg.CPU	0.024s	0.109s	1.106s	0.024s	0.109s	1.106s
	Avg.Dev.Ub	0.71%	0.36%	0.18% (1)	0.61%	0.36%	0.18% (1)
	Avg.Dev.Lb	11.45%	10.95%	10.68%	11.31%	10.95%	10.68%
	popsize	30	90	480	25	90	480
	nr_sub	0	0	0	2	0	0
J90	Avg.CPU	0.037s	0.173s	1.815s	0.037s	0.173s	1.815s
	Avg.Dev.Ub	0.89%	0.46%	0.145% (4)	0.77%	0.46%	0.145% (4)
	Avg.Dev.Lb	10.99%	10.35%	9.90%	10.80%	10.35%	9.90%
	popsize	20	75	440	15	75	440
	nr_sub	0	0	0	2	0	0
J120	Avg.CPU	0.055s	0.270s	2.992s	0.055s	0.270s	2.992s
	Avg.Dev.Ub	2.63%	1.47%	0.52% (10)	2.26%	1.41%	0.45% (16)
	Avg.Dev.Lb	34.19%	32.34%	30.82%	33.55%	32.18%	30.69%
	popsize	20	75	360	18	45	240
	nr_sub	0	0	0	3	3	3
RG300	Avg.CPU	0.335s	1.634s	16.361s	0.335s	1.634s	16.361s
	Avg.Dev.Ub	1.89%	1.09%	0.28%	1.45%	0.72%	0.00%
	Avg.Dev.Lb	830.02%	821.80%	812.97%	824.60%	817.36%	809.93%
	popsize	16	45	280	16	40	260
	nr_sub	0	0	0	8	7	5

Table 6. Average deviations (%) from the optimal makespan for J30.

	Maximum no. of schedules		
Algorithm	1,000	5,000	50,000
Kochetov and Stolyar (2003)	0.10	0.04	0.00
Debels et al. (2006)	0.27	0.11	0.01
DBGA	**0.12**	**0.04**	**0.02**
GA	**0.15**	**0.04**	**0.02**
Valls et al. (2002)	0.27	0.06	0.02
Alcaraz and Maroto (2001)	0.33	0.12	—
Alcaraz et al. (2004)	0.25	0.06	0.03
Valls et al. (2005)	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
Klein (2000)	0.42	0.17	—
Valls et al. (2005)	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 et al. (1998)	0.86	0.44	—
Kolisch and Drexl (1996)	0.74	0.52	—
Hartmann (1998)	1.03	0.56	0.23
Kolisch (1996b)	0.83	0.53	0.27
Coelho and Tavares (2003)	0.81	0.54	0.28
Kolisch (1995)	1.44	1.00	0.51
Hartmann (1998)	1.38	1.12	0.88
Kolisch (1996a, b)	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 7. Average deviations (%) from the critical-path-based lower bound for J60.

	Maximum no. of schedules		
Algorithm	1,000	5,000	50,000
DBGA	**11.31**	**10.95**	**10.68**
GA	**11.45**	**10.95**	**10.68**
Debels et al. (2006)	11.73	11.10	10.71
Valls et al. (2002)	11.56	11.10	10.73
Kochetov and Stolyar (2003)	11.71	11.17	10.74
Valls et al. (2005)	12.21	11.27	10.74
Alcaraz et al. (2004)	11.89	11.19	10.84
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	—
Klein (2000)	12.77	12.03	—
Nonobe and Ibaraki (2002)	12.97	12.18	11.58
Valls et al. (2005)	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, b)	13.66	13.21	—
Coelho and Tavares (2003)	13.80	13.31	12.83
Kolisch (1996b)	13.59	13.23	12.85
Baar et al. (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

problems for which $pt_2 - pt_1$ is equal to $2/4 \cdot makespan$. The procedure focuses iteratively on a later part of the schedule as follows: $[pt_1, pt_2]$ equals $[0, 1/2 \cdot makespan]$ for the first subproblem, $[1/4 \cdot makespan, 3/4 \cdot makespan]$ for the second subproblem, and $[1/2 \cdot makespan, makespan]$ for the third subproblem.

We tested the GA by fixing *nr_sub* at zero and the DBGA by fine-tuning *nr_sub* to an optimal value. To illustrate that the DBGA can level up the performance of the GA, it is impossible to use the number of generated schedules as a stop condition because considering a schedule for a subproblem 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 $pt_2 - pt_1$. Therefore, we use the number of generated schedules and a time equivalent as a stop criterion for the GA and the DBGA, respectively. More precisely, the time needed to solve a problem instance with the GA within a given schedule limit determines the maximal allowable time to solve the same instance with DBGA. In our experiments, we used an equal value of *popsize* for all iterations of the DBGA. Because an optimal value for *popsize* depends on the number of generated schedules (cf. supra), we allocate the available time for each iteration proportional to the value of $pt_2 - pt_1$. In doing so, we generate a more or less equal number of schedules in each iteration. In Figure 11, the percentages of the time allocations are depicted between brackets.

For our experiments, we imposed a stop condition of 1,000, 5,000, and 50,000 schedules (GA) or the corresponding time equivalent (DBGA). We tested the GA and the DBGA on the PSPLIB-data sets and on the RG300 data set, and the results can be found in Table 5. The line Avg.CPU reports the average CPU time needed to solve a problem instance. As the DBGA uses a time equivalent for the CPU time needed by the GA, the values for Avg.CPU are equal for GA and DBGA. The line Avg.Dev.Ub shows the average deviation from the best solutions found so far. For the RG300 instances, we used the solutions found within a time limit of 50,000 schedules and optimal values for *popsize* and *nr_sub*. For the PSPLIB data sets, these solutions can be found on http://www.bwl.uni-kiel.de/bwlinstitute/Prod/psplib/datasm.html. The number of problem instances for which the obtained solution is better than the best solution so far is indicated in parentheses in the same line (only displayed when positive). The line Avg.Dev.Lb gives the average deviation from the critical path-based lower bound.

Table 8. Average deviations (%) from the critical-path-based lower bound for J120.

	Maximum no. of schedules		
Algorithm	1,000	5,000	50,000
DBGA	**33.55**	**32.18**	**30.69**
GA	**34.19**	**32.34**	**30.82**
Valls et al. (2002)	34.07	32.54	31.24
Alcaraz et al. (2004)	36.53	33.91	31.57
Debels et al. (2006)	35.22	33.10	31.57
Valls et al. (2005)	35.39	33.24	31.58
Kochetov and Stolyar (2003)	34.74	33.36	32.06
Valls et al. (2005)	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 et al. (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 et al. (2005)	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, b)	39.65	38.77	—
Hartmann (1998)	45.82	42.25	38.83
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

Finally, the lines *popsize* and *nr_sub* report the optimal values for both parameters.

The test results of Table 5 can be interpreted as follows. First, the table reveals that the introduction of the DBGA has a beneficial effect for large problem instances. The percentage improvement (Avg.Dev.Ub) for the J30, J60, and J90 instances is small and only occurs for a stop criterion of 1,000 schedules. However, the large instances (J120 and RG300) can be improved by the DBGA, regardless of the stop criterion. As an example, the Avg.Dev.Ub for the RG300 instances decreases from 1.89% to 1.45% due to the decomposition of the main problem into eight subproblems. Second, the table reveals that *nr_sub* is negatively related to the stop criterion, i.e., the more schedules generated, the less beneficial the DBGA. Last, the results reveal that the *popsize* parameter is positively related to the stop criterion and negatively related to the size of the problem instance. Consequently, the use of a large population avoids the creation of a homogeneous population, and this effect becomes more important for small problem instances and high stop conditions. Note also that when $nr\_sub > 0$ in the DBGA, the corresponding optimal value for *popsize* slightly decreases compared to the *popsize* in the GA.

4.3. Comparative Computational Results

In this section, we illustrate that GA and DBGA are very effective heuristic procedures. 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 data sets 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 6, 7, and 8, we compare the results for GA and DBGA in Table 5 with these results for the data sets 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 and 1,000 schedules are used as a tiebreaker.

5. Conclusions

In this paper, we developed a new decomposition-based GA for the well-known resource-constrained project-scheduling problem. We developed a competitive GA and embedded this metaheuristic in a heuristic decomposition approach, resulting in the DBGA. The DBGA solves subproblems by iteratively focusing on a different subpart of the schedule. We performed detailed computational results on two problem sets: the well-known PSPLIB data set and a self-made RG300 data set. By using a schedule limit as a stop condition for the GA and a time equivalent for the DBGA, we are able to compare the performance of both procedures. The results indicate that the DBGA can improve the results of the GA (Table 5), especially for large problem instances (J120 and RG300) or low stop conditions (1,000 schedules). Experiments on the PSPLIB data set revealed that, in general, the GA and the DBGA outperform all other state-of-the-art procedures (Tables 6, 7, and 8).

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

References

Alcaraz, J., C. Maroto. 2001. A robust genetic algorithm for resource allocation in project scheduling. *Ann. Oper. Res.* **102** 83–109.

Alcaraz, J., C. Maroto, R. Ruiz. 2004. Improving the performance of genetic algorithms for the RCPS problem. *Proc. Ninth Internat. Workshop Project Management and Scheduling*. Nancy, France, 40–43.

Baar, T., P. Brucker, S. Knust. 1998. Tabu-search algorithms and lower bounds for the resource-constrained project scheduling problem. S. Voss, S. Martello, I. H. Osmon, eds. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Norwell, MA, 1–8.

Bouleimen, K., H. Lecocq. 2003. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *Eur. J. Oper. Res.* **149** 268–281.

Brucker, P., S. Knust, A. Schoo, O. Thiele. 1998. A branch & bound algorithm for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **107** 272–288.

Brucker, P., A. Drexl, R. Möhring, K. Neumann, E. Pesch. 1999. Resource-constrained project scheduling: Notation, classification, models and methods. *Eur. J. Oper. Res.* **112** 3–41.

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

Debels, D., B. De Reyck, R. Leus, M. Vanhoucke. 2006. A hybrid scatter-search/electromagnetism meta-heuristic for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **169**(3) 638–653.

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

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

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

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

Hartmann, S. 2002. A self-adapting genetic algorithm for project scheduling under resource constraints. *Naval Res. Logist.* **49** 433–448.

Hartmann, S., R. Kolisch. 2000. Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **127** 394–407.

Herroelen, W., E. Demeulemeester, B. De Reyck. 1999. A classification scheme for project scheduling. J. Weglarz, ed. *Project Scheduling—Recent Models, Algorithms and Applications. International Series in Operations Research and Management Science*, Vol. 14. Kluwer Academic Publishers, Boston, MA, 77–106.

Herroelen, W., B. De Reyck, E. Demeulemeester. 1998. Resource-constrained project scheduling: A survey of recent developments. *Comput. Oper. Res.* **25**(4) 279–302.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI.

Icmeli, O., S. S. Erenguc, C. J. Zappe. 1993. Project scheduling problems: A survey. *Internat. J. Oper. Productions Management* **13**(11) 80–91.

Kochetov, Y., A. Stolyar. 2003. Evolutionary local search with variable neighbourhood for the resource constrained project scheduling problem. *Proc. 3rd Internat. Workshop of Comput. Sci. Inform. Tech.*, Ufa, Russia.

Kolisch, R. 1995. *Project Scheduling Under Resource Constraints—Efficient Heuristics for Several Problem Classes.* Physica-Verlag, Heidelberg, Germany.

Kolisch, R. 1996a. Efficient priority rules for the resource-constrained project scheduling problem. *J. Oper. Management* **14** 179–192.

Kolisch, R. 1996b. Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. *Eur. J. Oper. Res.* **43** 23–40.

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

Kolisch, R., S. Hartmann. 1999. Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. J. Weglarz, ed. *Project Scheduling—Recent Models, Algorithms and Applications.* Kluwer Academic Publishers, Boston, MA, 147–178.

Kolisch, R., S. Hartmann. 2004. Experimental investigation of heuristics for resource-constrained project scheduling: An update. Working paper, Technical University of Munich, Munich, Germany.

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

Kolisch, R., A. Sprecher. 1997. PSPLIB—A project scheduling library. *Eur. J. Oper. Res.* **96** 205–216.

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

Li, K. Y., R. J. Willis. 1992. An iterative scheduling technique for resource-constrained project scheduling. *Eur. J. Oper. Res.* **56** 370–379.

Mausser, H. E., S. R. Lawrence. 1997. Exploiting block structure to improve resource-constrained project schedules. F. Glover, I. Osman, J. Kelley, eds. *Metaheuristics 1995: State of the Art.* Kluwer.

Merkle, D., M. Middendorf, H. Schmeck. 2002. Ant colony optimization for resource constrained project scheduling. *IEEE Trans. Evolutionary Comput.* **6**(4) 333–346.

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

Nonobe, K., T. Ibaraki. 2002. Formulation and tabu search algorithm for the resource constrained project scheduling problem (RCPSP). C. C. Ribeiro, P. Hansen, eds. *Essays and Surveys in Meta-Heuristics.* Kluwer Academic Publishers, Boston, MA, 557–588.

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

Palpant, M., C. Artigues, P. Michelon. 2004. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Ann. Oper. Res.* **131** 237–257.

Schirmer, A. 2000. Case-based reasoning and improved adaptive search for project scheduling. *Naval Res. Logist.* **47** 201–222.

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

Sprecher, A. 2002. Network decomposition techniques for resource-constrained project scheduling. *J. Oper. Res. Soc.* **53**(4) 405–414.

Tormos, P., A. Lova. 2001. A competitive heuristic solution technique for resource-constrained project scheduling. *Ann. Oper. Res.* **102** 65–81.

Tormos, P., A. Lova. 2003a. An efficient multi-pass heuristic for project scheduling with constrained resources. *Internat. J. Production Res.* **41** 1071–1086.

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

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

Valls, V., F. Ballestín, S. Quintanilla. 2004. A population-based approach to the resource-constrained project scheduling problem. *Ann. Oper. Res.* **131** 305–324.

Valls, V., F. Ballestin, S. Quintanilla. 2005. Justification and RCPSP: A technique that pays. *Eur. J. Oper. Res.* **165**(2) 375–386.

Valls, V., S. Quintanilla, F. Ballestín. 2003. Resource-constrained project scheduling: A critical activity reordering heuristic. *Eur. J. Oper. Res.* **149** 282–301.