

Genetic Algorithm for the  
Resource-Constrained Project Scheduling  
Problem Using Encoding with Scheduling  
Mode \*

Vu Thien Can,  
Department of Mathematics and Computer Science,  
University of Ho Chi Minh City, Vietnam

Jacques A. Ferland,  
Département d'informatique et de recherche opérationnelle,  
Université de Montréal, Canada

Nguyen Huu Anh,  
Department of mathematics and Computer Science,  
University of Ho Chi Minh City, Vietnam

December 6, 2004

---

\*This research was supported by NSERC grant (OGP 0008312) from Canada.

## Abstract

In this paper we use genetic algorithms (*GA*) to deal with the Resource-Constrained Project Scheduling Problem (*RCPSP*). We introduce a priority value encoding (representation) to implement the *GA* for *RCPSP*. First, we follow up the approach of Alcaraz and Maroto [1] to introduce an additional component in the encoding to indicate the scheduling mode (forward or backward) used to generate the corresponding schedule. Furthermore, the numerical results indicate that our method is slightly better as far as solution quality is concerned and requires smaller solution time than the *GA* method proposed by Alcaraz and Maroto in [1] where an activity list encoding is used.

**Key words:** genetic algorithm, resource-constrained project scheduling, metaheuristic.

## 1 Introduction

The Resource-Constrained Project Scheduling Problem (*RCPSP*) is a classical well-known problem where the activities of a project must be scheduled to minimize its makespan [7, 17, 22]. Several authors have suggested different methods to solve this problem. They can be classified into three categories. The exact methods [8, 9, 11, 26, 27] generate optimal solutions for small size problems (up to 60 activities). The heuristic procedures are the alternative to deal with larger problems [3, 12, 20, 21, 24]. These methods are serial or parallel schedule generation schemes. The third category includes metaheuristic methods like Tabu search [2, 28], simulated annealing [4, 5, 6], and genetic algorithm [1, 14, 19]. Recent numerical comparisons of these methods [1, 5, 15, 22] indicate that the three most efficient procedures are (in order) the genetic algorithms of Alcaraz and Maroto [1], the simulated annealing method of Bouleimen and Lecocq [5, 6], and the genetic algorithm of Hartman [14].

To improve Hartman genetic algorithm [14] using an activity list encoding (representation) of the solution, Alcaraz and Maroto [1] introduce an additional component in the encoding to indicate the scheduling mode (forward or backward) used to generate the corresponding schedule. The numerical results in [1] show the advantage of including the scheduling mode in the encoding. In this paper, to follow up this approach we present genetic algo-

rithms using priority value encoding with scheduling mode. The numerical results indicate that the variant using an encoding with scheduling mode generates better solutions than the variant where the encoding does not include the scheduling mode. Furthermore, the results indicate that our genetic algorithm is slightly better as far as solution quality is concerned and requires smaller solution time than the method proposed by Alcaraz and Maroto[1].

In Sections 2 and 3 we briefly summarize the descriptions of the RCPSP and of the genetic algorithm (*GA*) [10, 13, 18], respectively. In Section 4 we specify the operators (selection, crossover, and mutation) and the culling strategy (used to generate the new current population) that we use to implement the activity list or permutation based *GAs*. Then Section 5 includes the modifications required to implement the *GAs* using activity list encoding with scheduling mode. Similarly we introduce the operators and their adapted versions for the priority value based *GAs* and their variants using priority value encoding with scheduling mode in Sections 6 and 7, respectively. Section 8 includes the numerical results and their analysis. Concluding remarks are included in Section 9.

## 2 Problem description

In a Resource-Constrained Project Scheduling Problem (*RCPSP*), we consider a project including  $n$  activities  $j = 1, 2, \dots, n$  to be scheduled under resource and precedence constraints. The time required to complete an activity  $j$  is specified in terms of an integer number  $d_j$  of periods. It is also assumed that once initiated, any activity is completed without interruption.

On the one hand, the precedence constraints are induced by technological requirements to impose that any activity  $j$  must be scheduled for execution after the completion of all its immediate predecessors included in a set  $P_j$ . On the other hand, each activity  $j$  requires  $r_{jk}$  units of resource  $k$ ,  $k = 1, 2, \dots, K$ , during each period of its execution. Hence resource constraints are specified to limit the number of units of each resource  $k$  used in each period  $t$  to the number  $A_{kt}$  of units available.

The problem is to determine a starting period for each activity in order to minimize the total duration (makespan) of the project while satisfying the precedence and resource constraints. Note that referring to Herroelen et al. classification in [16], this problem is denoted as  $m/1/cpm/C_{\max}$ , and referring Brucker et al. notation in [7], as  $PS/prec/C_{\max}$ .

### 3 Genetic algorithm (*GA*) for *RCPSP*

Genetic algorithms (*GA*) [10, 13, 18] are population based techniques where an evolutionary mechanism that mimics a biological mechanism based on selection and natural evolution principles, is used to transform a population (set) of solutions from one iteration (generation) to the next in order to lead the search into promising regions of the feasible domain. An initial population  $G_0$  of  $m$  solutions is generated, and at each iteration, the size of the population remains the same. At each iteration three operators are applied to generate a set of new offspring solutions : a selection operator, a crossover operator, and a mutation operator.

The selection operator is used to select an even number of (parent) solutions from the current population. Each parent-solution is selected according to some strategy. The parent-solutions are paired, and a crossover operator is applied with some probability  $p_{cross}$  to each pair of parent-solutions to generate new feasible offspring-solutions inheriting interesting components contained in the parent-solutions. Then, a mutation operator is applied to modify the components of each offspring-solution with a small probability  $p_{mut}$ . The purpose of using such an operator is to promote diversity in the current population of solutions. Finally, a culling strategy is used to determine a new population by selecting among the parent-solutions and the offspring-solutions. The procedure is repeated until some stopping criterion is reached.

Now an essential feature of this solution approach is the encoding or representation of phenotype solutions into genotype vectors to which the different operators are applied. Several different encodings exist for the *RCPSP*, but we limit our presentation to the activity list (permutation-based) encoding given in Hartmann [14] and Alcaraz and Maroto [1], and to a priority value encoding quite similar to the priority value based encoding in Hartmann [14]. We also consider the corresponding encoding with scheduling mode as introduced by Alcaraz and Maroto [1] for the activity list encoding .

### 4 Activity list or permutation based *GA*

Our presentation borrows heavily from [1]. In this first encoding, each individual in the population (genotype) is represented as a vector

$$[j_1, j_2, \dots, j_n]$$

Figure 1: Project example having 8 activities.

corresponding to a permutation of the activities of the project. This permutation satisfies the precedence constraints; i.e. for any index  $j_i$ ,  $i = 2, 3, \dots, n$ ,

$$P_{j_i} \subset \{j_1, j_2, \dots, j_{i-1}\}.$$

Each genotype vector is decoded into a unique phenotype schedule according to the following serial scheduling scheme. Activity  $j_1$  is scheduled to start at time 0 (i.e. in the first period). Activity  $j_i$  is the  $i^{th}$  activity scheduled as early as possible after its predecessors according to the resources available.

To illustrate the serial scheduling scheme, consider the example illustrated in Figure 1 taken from [1] where the project includes 8 activities requiring one type of resource. Assume that  $A_{1t} = 8$  units of resource are available in each period  $t$ . With each activity-node  $j$ , we associate the pair  $(d_j, r_{j1})$ . The genotype individual

$$[1, 2, 3, 4, 5, 6, 7, 8]$$

translates into the unique feasible (phenotype) schedule illustrated in figure 2. It is interesting to note that the schedule in figure 2 corresponds also to the genotype individual

$$[2, 1, 4, 3, 5, 7, 6, 8]$$

In [1] the genotype individuals in the initial population are generated randomly as follows. Starting with an empty vector, the next activity is

Figure 2: Schedule with makespan of 14.

selected among those having all their predecessors already included in the vector according to a probability biased by the *LFT* rule.

Alcaraz and Maroto compare several selection operators [1], and the numerical results indicate that the best one is the 2-tournament operator where each parent solution is selected according to the following mechanism : two individual solutions are selected randomly from the current population, and the best fitted is selected as a parent-solution.

In this paper we compare numerically two different *GAs* variants using the one-point and the two-points crossover operators, respectively. The best results in [1] are obtained using these crossover operators. To specify these operators, denote the parent-solutions by  $PS^1$  and  $PS^2$ , and the offspring solution by  $OS^1$  and  $OS^2$ :

$$PS^1 : [ps_1^1, ps_2^1, \dots, ps_n^1]$$

$$PS^2 : [ps_1^2, ps_2^2, \dots, ps_n^2].$$

In the one-point crossover operator, an integer number  $q$  is selected randomly in the set  $\{1, 2, \dots, n-1\}$ . Then the first  $q$  components of  $os^1$  are the corresponding components of  $ps^1$  (i.e.,  $os_j^1 = ps_j^1, j = 1, 2, \dots, q$ ). The rest

of the activities  $os_{q+1}^1, os_{q+2}^1, \dots, os'_n$  are selected in the order in which they appear in the second parent-solution  $PS^2$ ; i.e., for  $j = q + 1, q + 2, \dots, n$ ,

$$os_j^1 = ps_k^2 \text{ where } k \text{ is the smallest index such that} \\ ps_k^2 \notin \{os_1^1, os_2^1, \dots, os_{j-1}^1\}.$$

The second offspring-solution  $OS^2$  is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

In the two-points crossover operator, two integer numbers  $q_1 < q_2$  are selected randomly in the set  $\{1, 2, \dots, n\}$ . Then the first  $q_1$  components of  $OS^1$  are the corresponding ones of  $PS^1$  (i.e.,  $os_j^1 = ps_j^1, j = 1, 2, \dots, q_1$ ). The components  $q_1 + 1, q_1 + 2, \dots, q_2$  are selected in the order in which they appear in the second-parent solution  $PS^2$ ; i.e. for  $j = q_1 + 1, q_1 + 2, \dots, q_2$ ,

$$os_j^1 = ps_k^2 \text{ where } k \text{ is the smallest index such that} \\ ps_k^2 \notin \{os_1^1, os_2^1, \dots, os_{j-1}^1\}.$$

Finally, the last components  $q_2 + 1, q_2 + 2, \dots, n$ , are selected in the order in which they appear in the first-parent solution  $PS^1$ ; i.e. for  $j = q_2 + 1, q_2 + 2, \dots, n$ ,

$$os_j^1 = ps_k^1 \text{ where } k \text{ is the smallest index such that} \\ ps_k^1 \notin \{os_1^1, os_2^1, \dots, os_{j-1}^1\}.$$

The second offspring-solution  $OS^2$  is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

For each pair of parent-solutions, the crossover operator is applied with a probability  $p_{cross}$ . If the operator is not applied, then the parent-solutions become their own offspring-solutions.

The mutation operator inducing the best results in [1] is an adaptation of Boctor's procedure [4] to generate neighbor solutions. Consider each activity sequentially. Determine randomly a new position in the genotype vector such that the precedence constraints are satisfied. Then the activity is moved to this new position with a probability  $p_{mut}$ .

Finally, the culling strategy is to replace the current population by the set of offspring-solutions to generate the new current population.

## 5 Activity list encoding with scheduling mode

In [1] Alcaraz and Maroto propose a variant of the activity list encoding where an additional component ( $n + 1$ ) of the vector is used to indicate the scheduling mode forward or backward ( $f/b$ ) used to decode the genotype vector into a (phenotype) schedule. The forward scheduling mode  $f$  is the serial scheduling scheme described before. The backward mode  $b$  is also a serial scheduling scheme where the last activity  $j_n$  is first scheduled. Activity  $j_{n-i+1}$  is the  $i^{th}$  activity scheduled as late as possible before its successors according to the of resources available.

The initial population is generated as before. To determine the scheduling mode  $f/b$  of each individual, we evaluate the makespans of the two schedules generated using the forward and the backward scheduling mode, respectively. The most fitted schedule (i.e., the schedule with the smallest makespan) induces the value  $f/b$  of the  $(n + 1)st$  component of the genotype vector.

The crossover operators are slightly modify to account for the scheduling mode. In the one-point crossover operator, if the first parent-solution  $PS^1$  mode is  $f$ , then  $OS^1$  is determined as before. Otherwise, if  $PS^1$  mode is  $b$ , then the last  $q$  components of  $OS^1$  are the corresponding components of  $ps^1$ ; i.e.,  $os_j^1 = ps_j^1, j = q + 1, \dots, n$ . The rest of the activities  $os_q^1, os_{q-1}^1, \dots, os_1^1$  are selected in the order in which they appear in  $PS^2$ ; i.e. for  $j = q, q - 1, \dots, 1$ ,

$$os_j^1 = ps_k^2 \text{ where } k \text{ is the largest index such that} \\ ps_k^2 \notin \{os_{j+1}^1, os_{j+2}^1, \dots, os_n^1\}.$$

Finally, the scheduling mode of  $OS^1$  is the same as  $PS^1$  (i.e.,  $os_{n+1}^1 = ps_{n+1}^1$ ). The second offspring-solution  $OS^2$  is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

In the two-points crossover, if the first parent-solution  $PS^1$  mode is  $f$ , then  $OS^1$  is determined as before. Otherwise, if  $PS^1$  mode is  $b$ , then the last  $(n - q_2)$  components of  $OS^1$  are the corresponding components of  $ps^1$  (i.e.,  $os_j^1 = ps_j^1, j = q_2 + 1, \dots, n$ ). The components  $q_1 + 1, \dots, q_2$  of  $OS^1$  are selected in the order in which they appear in the second parent-solution  $PS^2$ ; i.e. for  $j = q_2, q_2 - 1, \dots, q_1 + 1$ ,

$$os_j^1 = ps_k^2 \text{ where } k \text{ is the largest index such that} \\ ps_k^2 \notin \{os_{j+1}^1, os_{j+2}^1, \dots, os_n^1\}.$$

The first  $q_1$  components of  $OS^1$  are selected in the order in which they appear in  $PS^1$ ; i.e. for  $j = q_1, \dots, 1$

$$os_j^1 = ps_k^1 \text{ where } k \text{ is the largest index such that}$$

$$ps_k^1 \notin \{os_{j+1}^1, os_{j+2}^1, \dots, os_n^1\}.$$

Finally, the scheduling mode of  $OS^1$  is the same as  $PS^1$  (i.e.,  $os_{n+1}^1 = ps_{n+1}^1$ ). The second offspring solution  $OS^2$  is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

The same Boctor's type mutation operator (applied to the first  $n$  components) and the same culling strategy are used in these  $GAs$ .

## 6 Priority value based $GA$

The priority value encoding presented in this section is quite similar to the one used in Hartmann [14] except that the values associated with the activities in the genotype vector are integer values in the set  $\{0, 1, \dots, n-1\}$ . The smaller is the value associated with an activity, the higher is the priority to schedule it.

The genotype vector is decoded into a phenotype schedule according to the following serial scheduling where the activities are scheduled sequentially. Each time a new activity is scheduled, it is selected as one with the highest priority (i.e., with the smallest component in the genotype vector) among those having all their predecessors already scheduled. This activity is scheduled as early as possible after all its predecessors according to the resources available. Referring to the example illustrated in figure 1, the genotype individual

$$[0, 1, 2, 3, 4, 5, 6, 7]$$

translates into the same (phenotype) schedule illustrated in figure 2. Note also that this schedule corresponds also to the genotype individual

$$[1, 0, 3, 2, 4, 6, 5, 7].$$

The first individual in the initial population is generated according to the *LFT* rule where the component of the activity having the largest latest finishing time is equal to  $(n-1)$ . The other individuals are generated randomly by assigning a random integer in  $\{0, 1, \dots, n-1\}$  to each component of the

genotype vector. Thus it is possible for more than one activity to have the same value in the vector.

In our implementation we use the following pseudo-elitist selection operator. Each parent-solution is selected randomly in a subset of the current population obtained by eliminating the 25% less fitted solutions and the 25% best fitted solutions; i.e., this subset includes 50% of the current population including the (so called) average best fitted solutions. Furthermore, in each pair of parent-solutions, they are selected to be different.

An advantage of this encoding is to allow using the standard crossover operators. In this paper, we compare the variants using the one-point and the uniform crossover operators, respectively. For the sake of completeness, recall that in the one-point crossover operator an integer number  $q$  is selected randomly in the set  $\{1, 2, \dots, n - 1\}$ . Then the first offspring-solution  $OS^1$  inherits its first  $q$  components from parent-solution  $PS^1$  and the rest of its components from parent-solution  $PS^2$ ; i.e.

$$os_j^1 = \begin{cases} ps_j^1 & j = 1, 2, \dots, q \\ ps_j^2 & j = q + 1, \dots, n. \end{cases}$$

The second offspring-solution is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

To implement the uniform crossover operator, we use a vector  $vb$  of dimension  $n$  where the value of each component is selected randomly to be 0 or 1. The first offspring  $OS^1$  is generated as follows : for each  $j = 1, 2, \dots, n$

$$os_j^1 = \begin{cases} ps_j^1 & \text{if } vb_j = 0 \\ ps_j^2 & \text{if } vb_j = 1. \end{cases}$$

The second offspring-solution  $OS^2$  is determined by interchanging the roles of  $PS^1$  and  $PS^2$ .

For each pair of parent-solutions, the crossover operator is also applied with a probability  $p_{cross}$ , and the parent-solutions are their own offspring-solutions if the operator is not applied.

We use the following mutation operator. For each offspring-solutions, the operator is applied with probability  $p_{mut}$ . To apply the operator, we first select randomly an element  $\alpha \in \{0, 1, \dots, n - 1\}$ . Then select randomly a component of the offspring-solution vector that has a value different from  $\alpha$ , and replace it by  $\alpha$ . Now, in the case where all components of the offspring

solution vector have a value equal to  $\alpha$ , then select randomly a component to be modified to the value  $\alpha + 1$ , if  $\alpha < n - 1$ , or to the value  $\alpha - 1$ , otherwise.

Finally the culling strategy uses a 2-tournament selection operator to generate the new current population. Each individual of the new current population is selected as follows : two solutions are selected randomly in the set including those in the current population and the offspring-solutions (i.e., the set includes  $2n$  solutions), and the best fitted is included in the new current population.

## 7 Priority value encoding with scheduling mode

For the priority value encoding we can also apply the approach proposed by Alcaraz and Moroto in [1] and summarized in Section 5 where an  $(n + 1)$  additional component of the genotype vector is used to indicate the scheduling mode forward or backward ( $f/b$ ) used to decode it. The forward scheduling mode  $f$  is the serial scheduling scheme introduced above. The scheduling scheme is modified as follows in the backward mode  $b$ : each time a new activity is scheduled, it is selected as one having the lowest priority among those having all their successors already scheduled. This activity is scheduled as late as possible before its successors according to the resources available.

The initial population is generated as before, and the scheduling mode of each solution is selected according to the fitness of the schedules generated.

The crossover operators (one-point and uniform) remain unchanged. The scheduling mode of  $OS^1$  is the one of  $PS^1$  (i.e.,  $os_{n+1}^1 = ps_{n+1}^1$ ) if the number of components that it inherits from  $PS^1$  is larger or equal to the number of components that it inherits from  $PS^2$ . The scheduling mode of  $OS^2$  is the complementary one of  $OS^1$  (i.e.,  $os_{n+1}^2 = b(f)$  if  $os_{n+1}^1 = f(b)$ ).

Finally, the same mutation operator and the same culling strategy are used in the corresponding  $GAs$ .

## 8 Numerical results

The numerical experimentation completed by Alcaraz and Maroto in [1] indicates that their  $GAs$  using the activity list encoding with scheduling mode

are the most efficient metaheuristic procedures when compared with simulated annealing [6], tabu search [2], adaptive sampling [21, 29] and other *GAs* [14, 25]. In this section we want to complete a similar experimentation to compare the *GAs* using priority value encoding with the *GAs* of Alcaraz and Moroto.

Like Alcaraz and Moroto in [1], we are using the standard instances *J30*, *J60* and *J120* generated with the problem generator developed by Kolisch et al. [23] that are available in the Project Scheduling Problem LIBrary PSPLIB (<http://www.bwl.uni-kiel.de/Prod/psplib/>). Each set *J30* and *J60* includes 480 instances, and the set *J120* includes 620 instances. For instances in *J30*, the optimal values are known. For the instances in *J60* and *J120*, best known upper bounds (best known solutions) are available. Furthermore, the numerical results are obtained using a Pentium IBM-compatible computer 598MHz, 128Mb Ram running under Windows XP.

Five different *GAs* are compared numerically. The first *GA* denoted *CFA* uses the priority value encoding (without scheduling mode) and the one-point crossover operator as described in Section 6. The other four *GAs* use encoding with scheduling mode. *AM1* et *AM2* denote the *GAs* proposed by Alcaraz and Moroto [1] as described in Section 5. They both use the activity list encoding with scheduling mode, and the one-point and two-points crossover operator, respectively. Finally, the last two methods denoted *CFA1* and *CFAU* are the methods introduced in Section 7 using the priority value encoding with scheduling mode, and the one-point and the uniform crossover operator, respectively. All these method are coded in Java.

In designing our numerical experimentation, we take advantage of the numerical results obtained by Alcaraz and Moroto [1] to fixe most of the parameters. Hence, as in [1] we use the two stopping rules allowing a small number of 20 iterations and an intermediate number of 50 iterations, respectively. But we also consider a third stopping rule allowing for a larger number of 100 iterations. Note that in [1] the first two stopping rules are specified in terms of number of schedules generated, but they are quite equivalent according to the population sizes  $m$  used. The parameters used are summarized in Table 1. The same parameters are used for the five methods compared.

In our experimentation, each instance is solved 20 times using the same initial population of solutions for each of the 20 runs, and we denote

$B\ val$  : the value of the best solution generated

$A\ val$  : the average value of the 20 solutions generated

$W\ val$  : the value of the worst solution generated.

Using these values, we compare the 5 methods according to the following three global criteria:

- i) Percentage of deviation from the best solution (for problems in  $J30$ ) or from the best known solution (for problems in  $J60$  and in  $J120$ ):

Stopping rule (number of iterations)	Population size $m$	Prob. crossover $p_{cross}$	Prob. mutative $p_{mut}$
20	50	0.8	0.05
50	100	0.8	0.01
100	200	0.9	0.01

Table 1: Parameters used.

Hence for each set of problems ( $J30$ ,  $J60$  and  $J120$ ) and for each stopping rule we compute

$Ave\ Bdev(\%)$  : the average percentage of deviation of the value of the best solution generated ( $B\ val$ ) from the value of the best or the best known solution of each problem, taken over the set of problems.

$Ave\ Adev(\%)$  : the average percentage of deviation of the average value ( $A\ val$ ) of the 20 solutions generated from the value of the best or the best known solution of each problem, taken over the set of problems.

$Ave\ Wdev(\%)$  : the average percentage of deviation of the value of the worst solution generated ( $W\ val$ ) from the value of the best or the best known solution of each problem, taken over the set of problems.

- ii) Number of instances where the value of the best known solution is achieved:

For each set of problems and for each stopping rule we compute:

$NB_1$ : the number of instances where the value of the best known solution is achieved at least once among the 20 solutions generated

$NB_{20}$ : the number of instances where the best known solution is achieved for all the 20 solutions generated.

$$OPT_1(\%) = \frac{NB_1}{(\text{number of problems in the set})} \cdot 100$$

$$OPT_{20}(\%) = \frac{NB_{20}}{(\text{number of problems in the set})} \cdot 100$$

- iii) Average solution time (*CPU*):

For each set of problems and for each stopping rule we compute

*Ave CPU(sec)*: average solution to solve one instance of the problem set once.

The numerical results for the different criteria are summarized in Tables 2, 3, and 4 for problems *J30*, *J60*, and *J120*, respectively. In these tables, a column and a row are associated with each method and with each criterion, respectively. Furthermore, each component of a table corresponding to a pair (criterion, method) includes three different entries associated with the three stopping rules where the number of iterations are 20, 50, and 100, respectively.

Note that referring to these tables, the four most important criteria are the following : *Ave ADev*(%),  $OPT_1$ (%),  $OPT_{20}$ (%) and *Ave CPU(sec)*. The interest in having the two additional criteria *Ave BDev*(%) and *Ave WDev*(%) is to indicate the average width of the interval including the values of the solutions for the different instances. As expected, for the three problems (*J30*, *J60*, *J120*) we observe that the width of this interval decreases in general with the number of iterations (stopping rule).

Let us first compare the results obtained with *CFA* and *CFA1* to evaluate the advantage of including the scheduling mode into the priority value encoding. The results in Tables 2, 3, and 4 clearly indicate the superiority of the method *CFA1* using scheduling mode. Now consider the results in Table 5 obtained from those in Tables 2, 3, and 4. In this table, for the stopping criterion with 100 iterations and for each problem set (*J30*, *J60*,

	<i>CFA</i>	<i>AM1</i>	<i>AM2</i>	<i>CFA1</i>	<i>CFAU</i>
<i>Ave</i>	0.397	0.265	0.254	0.229	0.447
<i>Bdev</i> (%)	0.205	0.122	0.188	0.101	0.217
	0.139	0.073	0.116	0.060	0.150
<i>Ave</i>	0.999	0.562	0.572	0.991	0.959
<i>Adev</i> (%)	0.526	0.387	0.549	0.387	0.548
	0.366	0.339	0.294	0.189	0.359
<i>Ave</i>	1.811	0.964	0.953	2.730	1.573
<i>Wdev</i> (%)	0.965	0.492	1.070	0.491	0.925
	0.728	0.632	0.498	0.332	0.602
<i>OPT</i> <sub>1</sub>	83.75	89.79	90.21	80.42	83.54
(%)	90.00	94.38	89.79	95.21	89.58
	92.71	96.25	94.79	96.88	92.71
<i>OPT</i> <sub>20</sub>	61.46	76.25	75.42	62.50	66.25
(%)	72.71	82.50	71.04	82.50	72.92
	75.83	78.83	82.50	86.88	80.42
<i>Ave CPU</i>	0.1	0.2	0.2	0.3	0.3
( <i>sec</i> )	0.6	0.8	0.8	0.6	0.6
	2.5	4	4	3	3

Table 2: Problems *J30*-global comparison

and *J120*) we indicate the percentage of improvement  $IAdev(\%)$  of the average of deviation  $Ave Adev(\%)$  from the value of the best known solution, the percentage of improvement  $IOPT_1(\%)$  in the number of problems where the value of the best known solution is achieved, and the percentage of increase in solution time  $ICPU(\%)$  when the method *CFA1* is used rather than *CFA*. The figures in Table 5 clearly indicate that the percentages of improvement in solution quality ( $IAdev(\%)$  and  $IOPT_1(\%)$ ) increase with the size of the problems while percentage of the increase in solution time ( $ICPU(\%)$ ) decreases. These results confirm those of Alcaraz and Maroto in [1] to indicate the advantage of using scheduling mode in encoding.

Comparing the two methods with priority value encoding with scheduling mode *CFA1* and *CFAU* (where the one-point and the uniform crossover operators are used, respectively), the results in Tables 2, 3, and 4 clearly indicate the superiority of *CFA1* over *CFAU* as far as solution quality is concerned even if the average solution time is the same for both methods.

	<i>CFA</i>	<i>AM1</i>	<i>AM2</i>	<i>CFA1</i>	<i>CFAU</i>
<i>Ave</i>	1.795	1.187	0.802	0.799	1.958
<i>Bdev</i> (%)	1.036	0.408	0.555	0.407	1.407
	0.524	0.143	0.386	0.131	0.698
<i>Ave</i>	2.857	2.146	2.146	1.381	3.023
<i>Adev</i> (%)	1.702	0.812	2.046	0.793	2.046
	1.289	0.483	1.015	0.509	1.358
<i>Ave</i>	4.101	3.050	1.874	1.924	4.028
<i>Wdev</i> (%)	2.466	1.270	11.893	1.284	2.913
	2.059	0.866	1.723	0.894	2.082
<i>OPT</i> <sub>1</sub>	67.29	76.46	84.17	85.63	69.38
(%)	73.96	88.96	86.88	88.33	72.50
	77.50	90.83	86.67	91.67	78.75
<i>OPT</i> <sub>20</sub>	55.21	58.75	68.96	68.96	58.13
(%)	63.54	73.13	38.75	72.50	64.58
	62.92	72.92	65.21	73.33	65.42
<i>Ave CPU</i>	0.6	0.7	0.7	0.5	0.5
( <i>sec</i> )	2	3	3	2.4	2.4
	8	13	13	9	9

Table 3: Problems *J60*-global comparison

Similarly, if we refer to Tables 2, 3, and 4 to compare the two methods with activity list encoding with scheduling mode *AM1* and *AM2* (where the one-point and the two-points crossover operators are used, respectively), we can conclude that *AM1* generates better results than *AM2* using similar solution time.

Now, we proceed to compare the methods *AM1* and *CFA1*. First, consider the average percentage of deviation *Ave Adev*(%). *CFA1* generates solutions with better *Ave Adev*(%) than *AM1* for all problems *J30*, *J60* and *J120* with the three stopping rules except for one case (*J30* with 20 iterations). *CFA1* is also better as far as the percentage *OPT*<sub>1</sub>(%) of instances where the best known solution is achieved at least once among the 20 solutions generated. Indeed *AM1* is better in only two cases (*J30* with 20 iterations and *J60* with 50 iterations). Similar conclusions hold when we consider the percentage *OPT*<sub>20</sub>(%).

Consider the results in Table 6 where for each problem we compare the

	<i>CFA</i>	<i>AM1</i>	<i>AM2</i>	<i>CFA1</i>	<i>CFAU</i>
<i>Ave</i>	4.938	1.424	1.857	1.413	5.380
<i>Bdev</i> (%)	0.819	0.348	0.439	0.338	2.481
	1.487	0.221	0.415	0.203	1.799
<i>Ave</i>	6.617	2.381	3.188	2.360	7.134
<i>Adev</i> (%)	3.136	0.957	1.114	0.915	3.580
	2.136	0.606	0.833	0.591	2.459
<i>Ave</i>	8.360	3.328	4.093	3.269	8.713
<i>Wdev</i> (%)	3.353	1.592	1.794	1.514	4.518
	2.793	0.977	1.255	0.958	3.105
<i>OPT</i> <sub>1</sub>	35.50	63.00	65.17	64.83	35.50
(%)	70.00	81.33	78.83	82.00	58.33
	70.83	81.83	79.83	83.50	68.67
<i>OPT</i> <sub>20</sub>	23.67	26.33	26.17	27.00	24.00
(%)	52.67	55.00	55.33	55.83	52.17
	68.17	69.17	69.67	69.17	67.17
<i>Ave CPU</i>	2	3	3	2.4	2.4
( <i>sec</i> )	18	14	15	10	10
	45	60	60	48	48

Table 4: Problems *J120*-global comparison

number of instances where a method has a better or as good average value *Aval* of the 20 solutions generated than the other. The first two columns are associated with methods *AM1* and *CFA1*, respectively, and a row is associated with each problem *J30*, *J60*, and *J120*. Each component of the Table corresponding to a pair (problem, method) includes three different entries associated with the tree stopping rules. The last column *DIF* correspond to the difference between the columns associated with *CFA1* and *AM1*, respectively. Hence when the value in this column is positive, this implies that *CFA1* is better than *AM1* with respect to this criterion, and vice versa. Referring to Table 6, we may conclude that *CFA1* is slightly better than *AM1* with respect to this criterion also. Furthermore, it is interesting to note the clear superiority of *CFA1* for the larger instances of *J120*.

Finally, the results in Tables 2, 3, and 4 indicate a quite clear dominance of *CFA1* over *AM1* as far as average solution time (*Ave CPU*) is concerned. Indeed, *AM1* requires a smaller average solution time only for this instances

	<i>J30</i>	<i>J60</i>	<i>J120</i>
<i>IAdev</i> (%)	48.36	58.92	72.33
<i>I OPT<sub>1</sub></i> (%)	4.30	15.46	15.81
<i>ICPU</i> (%)	16.67	11.11	6.25

Table 5: Percentage of improvement when using scheduling mode for the stopping criterion with 100 iterations

	<i>AM1</i>	<i>CFA1</i>	<i>DIF</i>
<i>J30</i>	437	337	-100
	441	447	6
	401	455	54
<i>J60</i>	320	447	127
	414	436	-22
	430	420	-10
<i>J120</i>	392	414	22
	462	480	18
	503	527	24

Table 6: Pairwise comparison of *AM1* and *CFA1* with respect to the criterion *A Dev*(%)

in *J30* when the stopping criterion is equal to 20 iterations. In all the other cases, the *Ave CPU* of *AM1* is at least 20% (and reach up to 30%) larger than for *CFA1*.

## 9 Conclusion

In this paper we follow up the nice idea due to Alcaraz and Maroto [1] of including the scheduling mode into the activity list encoding of solutions to solve the *RCPSP* with *GAs*. Our method use priority value encoding rather than activity list encoding. The numerical results indicate the superiority of our method using scheduling mode (*CFA1*) over the one (*CFA*) without scheduling mode. This conclusion is consistent with the results in [1]. Then for the method of Alcaraz and Maroto, the numerical results indicate that, for the values of the parameters used, a one-point crossover operator (*AM1*)

seems more efficient than a two-points crossover operator ( $AM2$ ). Similarly, for our method, a one-point crossover operator ( $CFA1$ ) seems more efficient than a uniform crossover operator ( $CFAU$ ). Finally, comparing  $AM1$  and  $CFA1$ , the numerical results indicate that  $CFA1$  is slightly better as far as solution quality is concerned, and that the solution time for  $CFA1$  is smaller, in general.

## References

- [1] J. Alcaraz, C. Maroto, A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, **102** (2001), 83–109.
- [2] T. Baar, P. Brucker, S. Knust, Tabu search algorithms for resource-constrained project scheduling problems, in S. Voss, S. Martello, I. Osman, C. Roucairol (Eds.), *Metaheuristics : Advances and Trends in Local Search Paradigms for Optimisation*, Kluwer, 1997, pp. 1–18.
- [3] F.F. Boctor, Some efficient multi-heuristic procedures for the resource-constrained project scheduling problem, *European Journal of Operational Research*, **49**(1), (1990), 3–13.
- [4] F.F. Boctor, Resource-constrained project scheduling by simulated annealing, *International Journal of Production Research*, **34** (8), (1996), 2335–2351.
- [5] K. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version, *European Journal of Operational Research*, **149** (2003), 268–281.
- [6] K. Bouleimen, H. Lecocq, A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem, in: G. Barbarosoglu, S. Karabati, L. Özdamar, C. Ulasoy (Eds), *Proceedings of the Sixth International Workshop on Project Management and Scheduling*, Bogaziçi University Printing Office, Istanbul, 1998, pp. 19–22.
- [7] P. Brucker, A. Drexl, R. Nöhring, K. Neumann, E. Pesch, Resource-constrained project scheduling : Notation, models and methods, *European Journal of Operational Research*, **112**(1), (1999), 262–273.
- [8] P. Brucker, S. Knust, A. Schoo, O. Thiele, A branch-and-bound algorithm for the resources-constrained project scheduling problem, *European Journal of Operational Research*, **107**, (1998), 272–288.
- [9] N. Christofides, R. Alvarez-Valdes, J.M. Tamarit, Project scheduling with resource constraints : A branch-and-bound approach, *European Journal of Operational Research*, **29**(2), (1987), 262–273.

- [10] L. Davis, *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York, 1991.
- [11] E. Demeulemeester, W. Herroelen, A branch-and-bound procedure for the multiple resource-constrained project scheduling problem, *Management Science*, **38**(12), (1992), 1803–1818.
- [12] E. Demeulemeester, W. Herroelen, New benchmark results for the resource-constrained project scheduling problem, *Management Science*, **43**(11), (1995), 1485–1492.
- [13] D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, Addison Wesley, 1991.
- [14] S. Hartmann, A competitive genetic algorithm for the resource-constrained project scheduling, *Naval Research Logistics*, **456**,(1998), 733–750.
- [15] S. Hartmann, R. Kolisch, Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem, *European Journal of Operational Research*, **127**, (2000), 394–407.
- [16] W. Herroelen, E. Demeulemeester, B. De Reick, A classification scheme for project scheduling, In : J. Weglarz (Ed.), *Project Scheduling : Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1999, pp. 1–26.
- [17] W. Herroelen, E. Demeulemeester, B. De Reyck, Resource-constrained project scheduling – A survey of recent developments, *Computers and Operations Research*, **25**(4), (1998), 279–302.
- [18] J.H. Holland, *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor, 1975.
- [19] U. Kohlmorgen, H. Schmeck, K. Haase, Experiences with fine-grained parallel genetic algorithms, *Annals of Operations Research*, **90**, (1999), 203–219.
- [20] R. Kolisch, Efficient priority rule for the resource-constrained project scheduling problem, *Journal of Operational Management*, **14**, (1996), 179–192.

- [21] R. Kolisch, A. Drexl, Adaptive search for solving hard scheduling problems, *Naval Research Logistics*, **43**, (1996), 23–40.
- [22] R. Kolisch, S. Hartmann, Heuristic algorithms for solving the resource-constrained project scheduling problem : Classification and computation analysis, In : J. Weglarz (Ed.), *Project Scheduling : Recent Models, Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1999, pp. 147–178.
- [23] R. Kolisch, A. Sprecher, A. Drexl, Characterisation and generation of a general class of resource-constrained project scheduling problem, *Management Science*, **41** (10), (1995), 1693–1703.
- [24] J.K. Lee, Y.D. Kim, Search heuristics for the resource-constrained project scheduling problem, *Journal of Operational Research Society*, **47**(1996), 678–689.
- [25] V.J. Leon, R. Balakrishnan, Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling, *ORSpektrum*, **17**, (1995), 173–182.
- [26] A. Mingozzi, V. Maniezzo, S. Riccardelli, L. Bianco, An exact algorithm for project scheduling with resources constraints based on a new mathematical formulation, *Management Science*, **44**, (1998), 714–729.
- [27] J.H. Patterson, A comparison of exact approaches for solving the multiple constrained resource project scheduling problem, *Management Science*, **30**(7), (1984), 854–867.
- [28] E. Pinson, C. Prins, F. Rullier, Using tabu search for solving the resource-constrained project scheduling problem, In : *Proceedings of the 4th International Workshop on Project Management and Scheduling*, Leuven, Belgium, 1994, pp. 102–106.
- [29] A. Schirmer, S. Riesenberger, Class-based control schemes for parametrized project scheduling heuristics, Working paper, Institut für Betriebswirtschaftslehre, Universität Kiel, Kiel, Germany (1998).