European Journal of Operational Research 205 (2010) 31–41

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem

Sonda Elloumi [a,*], Philippe Fortemps [b]

[a] Research unit Logistique, Gestion Industrielle et de la Qualité, Institut Supérieur de Gestion Industrielle, Sfax, Tunisia
[b] Service de Mathématique et Recherche Opérationnelle, Faculté Polytechnique, Université de Mons, Belgium

A R T I C L E   I N F O

Article history:
Received 11 June 2007
Accepted 9 December 2009
Available online 21 December 2009

Keywords:
Project scheduling
Resource-constrained
Multiple modes
Evolutionary algorithms
Bi-objective approach
Clustering

A B S T R A C T

We consider the multi-mode resource-constrained project scheduling problem (MRCPSP), where a task has different execution modes characterized by different resource requirements. Due to the nonrenewable resources and the multiple modes, this problem is NP-hard; therefore, we implement an evolutionary algorithm looking for a feasible solution minimizing the makespan.

In this paper, we propose and investigate two new ideas. On the one hand, we transform the problem of single objective MRCPSP to bi-objective one to cope with the potential violation of nonrenewable resource constraints. Relaxing the latter constraints allows to visit a larger solution set and thus to simplify the evolutionary operators. On the other hand, we build the fitness function not on a priori grid of the bi-objective space, but on an adaptive one relying on clustering techniques. This proposed idea aims at more relevant fitness values.

We show that a clustering-based fitness function can be an appealing feature in multi-objective evolutionary algorithms since it may promote diversity and avoid premature convergence of the algorithms. Clustering heuristics require certainly computation time, but they are still competitive with respect to classical niche formation multi-objective genetic algorithm.

1. Introduction

One of the most studied generalizations of the resource-constrained project scheduling problem (RCPSP) is the multi-mode resource-constrained project scheduling problem (MRCPSP). It is closer to real problems than RCPSP since each activity[1] may be performed in one out of several modes and it deals with more than one kind of resources. Noticeably, the MRCPSP is more complex than the RCPSP, which is itself NP-hard. Sprecher and Drexl (1998) prove that highly resource-constrained projects with at least 20 activities and 3 modes per activity cannot be solved by exact optimization procedures within a reasonable computation time. Furthermore, Kolisch and Drexl (1997) demonstrate that, if at least 2 nonrenewable resources are taken, the problem of finding a feasible solution for the MRCPSP is NP-complete.

According to Boctor (1996a,b), the pioneer studies of the MRCPSP have been carried out by Elmaghraby (1977), Słowiński and Węglarz (1978) and Węglarz (1980). These authors have studied the preemptive case. Słowiński (1981) also tackles the preemptive MRCPSP, but under multiple criteria, such as makespan, maximum lateness, and cost criteria. Then, Talbot (1982) tackles the nonpreemptive case to give a mathematical formulation (on which many post-studies are based, like Sprecher (1994), Sprecher et al. (1997) and Hartmann (2001), etc.).

Talbot (1982) proposes a deterministic enumeration scheme. Analogously, Patterson et al. (1989, 1990) proposes a backtracking algorithm. Other exact procedures have been proposed, e.g. by Hartmann and Drexl (1998), Sprecher and Drexl (1998) and Sprecher et al. (1997). These are different variants of Branch-and-Bound method.

Heuristic methods have been conceived to solve the MRCPSP. Talbot (1982) proposes a procedure to heuristically solve large-sized problems. Boctor (1993) presents a comparison of 21 heuristic scheduling rules. Boctor (1996a) proposes a new efficient heuristic algorithm. The latter is a particular heuristic because it allows scheduling more than one activity at a time. Kolisch and Drexl (1997) conceive a local search procedure. The method begins by constructing an initial solution. Then an improvement of the mode assignment is performed. And finally, the best schedule is determined on the basis of the best mode assignment. Bianco et al. (1998) treat the MRCPSP, but they restrict the nonrenewable resource constraints to the unique budgetary constraint. Drexl and Grüenewald (1993) propose a stochastic scheduling method.

* Corresponding author. Tel.: +216 22 57 88 90; fax: +216 74 460 744.
E-mail addresses: sonda.elloumi@fsegs.rnu.tn, elloumi_sonda@yahoo.fr (S. Elloumi), philippe.fortemps@umons.ac.be (P. Fortemps).

[1] In this paper, we use the expression jobs and activities synonymously.

Słowiński et al. (1994) propose a decision support system for multi-objective MRCPSP based on three heuristics, namely: parallel priority rules, Simulated Annealing and Branch-and-Bound.

Several metaheuristic procedures have been devoted to MRCPSP. Damak et al. (2009) solve the problem with a differential evolution algorithm; whereas Jarboui et al. (2008) apply a particle swarm algorithm. Józefowska et al. (2001) present two versions of simulated annealing approach: with and without penalty function. Likewise, Bouleimen and Lecocq (2003) conceive a simulated annealing algorithm for the RCPSP and its multi-mode version. Ranjbar et al. (2009) propose a hybrid scatter search algorithm. Özdamar (1999) conceive a hybrid genetic algorithm for the MRCPSP; the main particularity of the algorithm appears in the solution encoding by the mode assignment and the priority rules. Hartmann (2001) proposes a genetic algorithm relying on two local search procedures already proposed by Sprecher et al. (1997). Hartmann (2001) uses the preprocessing procedure and the local search heuristic applied on individuals that violate the nonrenewable resource constraints. In the fitness function he proposes, solutions satisfying the nonrenewable resource constraints (i.e. nonrenewable resource feasible individuals) have been assigned their makespan; whereas the others have their fitness equal to the makespan upper bound added to a penalty value. Finally, the GA is enhanced with a left shift procedure. Alcaraz et al. (2003) and Elloumi et al. (2006) apply a similar framework, trying to overcome Hartmann's GA drawbacks. On the one hand, they have proposed new fitness functions; on the other hand, they have not employ the multi-mode left shift applied by Hartmann.

After this brief survey of literature, showing arguments in favor of metaheuristic approaches to MRCPSP, we propose in this paper a new evolutionary algorithm (EA) to cope with nonpreemptive MRCPSP, where the objective is the minimization of the project duration. The satisfaction of the nonrenewable resource constraint is relaxed into an additional penalty objective to be minimized. Therefore, the original single objective problem is transformed into a bi-objective one. In order to avoid premature convergence in this bi-objective search space, the EA is enhanced by the use of an adaptive grid, relying on clustering techniques. The latter issue is quite innovating and appears to be promising in general multi-objective combinatorial optimization.

The remainder of this paper is organized as follows; Section 2 is devoted to the presentation of the problem. The third section deals with the new EA steps and its features, and among others, the clustering-based adaptive grid. The last section is oriented towards the analysis of the algorithm computational results; we namely provide results drawn from the comparison of our EA with other metaheuristics proposed in MRCPSP literature.

2. Problem description

De Reyck and Herroelen (1999) proposed an overview of the most important project scheduling problem types found in the body of literature. Problems are classified according to the precedence relation type, the number of modes and both number and kind of resources considered. The MRCPSP is characterized by strict precedence relations, multiple execution modes and multiple renewable as well as nonrenewable resource types.

We consider a project consisting of J activities labeled $j = 1,\dots,J$. There are precedence relations between some of them. These precedence relations induce a partial order among the activities. We extend this partial order into a linear order (often called "topological order"), i.e. the activities are numerated such that different activities enjoy different numbers and an activity has always a label greater than all its predecessors. The precedence relations can be depicted by an acyclic activity-on-node network. We set additional activities $j = 0$ and $j = J + 1$ which represent, respectively, unique dummy source and sink nodes. Three most used categories of resources are considered:

- Renewable resources (whose set is referred to by $\mathcal{K}^{\rho}$): the per-period availability of a renewable resource k is denoted by R_k^{ρ};
- Nonrenewable resources (whose set is referred to by $\mathcal{K}^{\upsilon}$): the of a resource $k \in \mathcal{K}^{\upsilon}$ is designated by R_k^{υ};
- Doubly constrained resources: we deal with this type of resources by representing it in both kinds of constraints: the renewable and nonrenewable resources.

Each activity may be executed in one out of several modes. A mode is a way of performing a job. It reflects, for the activity in question, first the consumption of each resource k belonging to either category of resources, and second the related duration. For each activity $j = 1,\dots,J$, we distinguish a set of different modes; the latter is denoted by $\mathcal{M}_j = \{1,\dots,M_j\}$. The job j performed on a mode $m \in \mathcal{M}_j$ has a processing time referred to by p_{jm}, it requires r_{jmk}^{ρ} units of each renewable resource $k \in \mathcal{K}^{\rho}$ in each period it is processed, and r_{jmk}^{υ} units of each nonrenewable resource $k \in \mathcal{K}^{\upsilon}$. Each dummy activity is supposed to have a unique mode characterized by no request on any resource and nil duration.

The objective of this study is to find a mode and a finish time for each activity so that the schedule is feasible with respect to the precedence and resource constraints, and the makespan of the project is minimal.

3. Proposed evolutionary algorithm

In this section, we describe our proposal of EA to solve the MRCPSP. In the following, we define the proposed algorithm stages and especially further proposed procedures particularizing the proposed EA. Fig. 1 depicts the proposed EA scheme.

3.1. Basic scheme

First of all, a preprocessing procedure is applied in project data to efficiently reduce the search space. After that, the EA starts with generating an initial population; the number of individuals in a population $\mathcal{POP}$ is constant through generations, say POP; we assume POP to be an even integer. As the problem of finding a feasible solution for the MRCPSP is NP-complete if at least two nonrenewable resources are considered, we find appropriate the introduction of infeasible solutions with respect to nonrenewable resources into the search space. The violation of these constraints is penalized. Then, for each individual the makespan and a penalty value will be jointly considered in the fitness calculation. Next, a local search procedure is applied on infeasible solutions to improve them by trying to reduce their penalties. $POP/2$ pairs of individuals are randomly selected to undergo the crossover step. The set $\mathcal{CHI}$ of resulting children undergoes mutation with a probability p_{mut}. Once again, the fitness function for the newly produced individuals is computed by considering both makespan and penalty as two criteria to be minimized. The problem comes to a bi-objective problem. A new approach using clustering heuristics is proposed to deal with the multi-objective problem. Here intervenes the selection operator to maintain POP individuals ready to contribute in the next generation. This procedure is repeated until the stopping criterion is reached.

3.2. Definition of individuals

The encoding chosen for individuals is an activity list and the related mode assignment representation. Obviously, activity finish

```
1. Generate initial population $\mathcal{POP}$ consisting of $POP$ chromosomes;
2. Apply a local search procedure to each individual of the population;
3. Initialize the current generation number to 0;
4. While generation number < GEN and run-time < CPU time limit do
    4.1. Increment the generation number;
    4.2. Create children
        4.2.1. Select $POP$ individuals for crossover;
        4.2.2. Produce a set $\mathcal{CHI}$ of $POP$ children from pairs of the selected individuals by crossover;
        4.2.3. Apply mutation to $\mathcal{CHI}$ with a probability $p_{mut}$;
    4.3. Compute fitness values by:
        4.3.1. Computing penalty values;
        4.3.2. Performing the rank-based fitness assignment method on the basis of two criteria:
               makespan and penalty;
        4.3.3. Applying a clustering heuristic preocedure (K-means or hierarchical) to compute density;
    4.4. Reproduce $\mathcal{POP}$ := selected individuals from $\mathcal{POP} \cup \mathcal{CHI}$.
5. Return the best found solution.
```

Fig. 1. Proposed EA.

times depend at the same time on the two above-mentioned individual elements. So, our EA deals simultaneously with both sequencing and mode assignment problems.

Like Hartmann (2001), the representation used is a pair $I = (\lambda, \mu)$ of two vectors. The first vector $\lambda = (j_1, \ldots, j_J)$ denotes a precedence feasible activity list. The second vector $\mu = (m_1, \ldots, m_J)$ symbolizes the modes relating to the corresponding jobs in the list, i.e. m_1 is the mode associated to the first activity in λ i.e. j_1, and so on.

3.3. Search space

Since the presence of nonrenewable resources affects the NP-completeness of the problem (cf. Kolisch and Drexl, 1997), we follow Hartmann (2001) and Alcaraz et al. (2003) in introducing into the search space the set of MRCPSP solutions disregarding the nonrenewable resource constraints. Therefore, our search space consists in solutions satisfying precedence constraints and renewable resource constraints; nonrenewable resource constraints may be violated. A penalty function $L(\mu)$ to be minimized enforces the satisfaction of the nonrenewable resource constraints – where μ is the vector of mode assignment. In fact, this measure depends only on the mode assignment because it is related to nonrenewable resource requirement, and not on the precedence relation nor on the renewable resource consumption; $L(\mu)$ calculation is explained in Section 3.6.

3.4. Preprocessing

This step is introduced before the beginning of the EA in order to reduce the search space. The reduction procedure was first developed by Sprecher et al. (1997) to accelerate their branch-and-bound algorithm for the MRCPSP. The idea behind this procedure is to exclude modes and/or renewable as well as nonrenewable resources from the input data. Because of the interdependency between the elimination of these contents, the latter are dealt with as follows:

Step 1: Remove all non-executable modes from the project data.
Step 2: Delete the redundant nonrenewable resources.
Step 3: Eliminate all inefficient modes.
Step 4: If any mode has been erased within *Step 3*, go to *Step 2*.

For more details of the components interdependency and the merit of the procedure, we refer the reader to Sprecher et al. (1997) and Hartmann (2001).

3.5. Initial population

Initial individuals are obtained by fixing the activities modes randomly from the remaining set of modes after the preprocessing procedure. Then, the activity list is constructed with respect to the resulting mode assignment. The ensuing schedule is precedence and renewable resource feasible, but it is not necessarily nonrenewable resource feasible.

The initial generation is obtained by repeating the next steps *POP* times. Firstly, a mode assignment is generated by randomly selecting $m(j) \in \mathcal{M}_j$ for activities $j = 1, \ldots, J$. Secondly, following Hartmann (2001), the resulting mode assignment is checked for nonrenewable resource feasibility. In presence of violation, a local search procedure is applied trying to improve the current mode assignment. The process consists in randomly selecting a job j which has more than one mode alternative, and its current mode is changed by randomly selecting $m'(j) \in \mathcal{M}_j \setminus \{m(j)\}$. The result is a new mode assignment m'. If there is improvement, $m(j)$ is replaced by $m'(j)$. This process is repeated until J consecutive unsuccessful trials to improve the mode assignment have been made or, in the best case, until the individuals become nonrenewable resource feasible. Thirdly, we adopt the mode assignment and construct a precedence feasible schedule by randomly sampling (cf. Kolisch, 1996). The procedure consists of J stages; at each stage, a decision (say also eligible) set is determined; this set includes all unscheduled activities which every predecessor is scheduled yet. An activity is chosen randomly and scheduled at its earliest precedence and renewable resource feasible start time. These steps are repeated until the J steps are fulfilled. Finally, activities finish times are computed and the makespan of the project is derived.

3.6. Fitness computation

Although we are dealing with a single performance measure, our fitness function behaves by considering an additional performance measure to be minimized. The latter takes its favor from introducing solutions which are infeasible with regard to nonrenewable resource constraints; it consists of a penalty value attributed to these solutions. The fitness of a given individual $I = (\lambda, \mu)$ is computed in the following manner – remind that $\lambda = (j_1, \ldots, j_J)$ and $\mu = (m_1, \ldots, m_J), m_i$ being the mode associated to j_i, the ith activity in λ.

Like Hartmann (2001), let us assign for each nonrenewable resource $k \in \mathcal{K}^\upsilon$ a penalty measure:

$$L_k(\mu) = R_k^\upsilon - \sum_{i=1}^{J} r_{j_i m_i k}^\upsilon;$$

$L_k(\mu)$ designates the leftover capacity of the nonrenewable resource $k \in \mathcal{K}^\upsilon$ related to the mode assignment μ. A negative value of $L_k(\mu)$ implies that the availability of the nonrenewable resource k is exceeded and, consequently, the mode assignment μ is infeasible with respect to the nonrenewable resource k on hand. Then, we propose an aggregation measure $L(\mu)$ given by:

$$L(\mu) = -\sum_{\substack{k \in \mathcal{K}^\upsilon \\ L_k(\mu) < 0}} \frac{L_k(\mu)}{R_k^\upsilon}.$$

That is, $L(\mu)$ considers only the negative $L_k(\mu)$; in other words, it takes into account only nonrenewable resources whose total requirement exceeds the capacity. $L(\mu)$ is determined by the sum of adjusted $L_k(\mu)$, which amounts to excess percentage of resource requirements. Indeed, each excess of resource requirement (i.e. $-L_k(\mu) > 0$) is divided by the related resource availability to obtain a relevant excess ratio expressed in percentage. While other authors (e.g. Hartmann, 2001 and Alcaraz et al., 2003) directly aggregate the resource excess, we prefer to normalize them w.r.t. their respective resource availability, since the latter can be very different. So, their aggregation is now consistent. Clearly,

$$L(\mu) \begin{cases} > 0 & \text{if there is any excess of an availability of} \\ & \text{nonrenewable resource;} \\ = 0 & \text{otherwise.} \end{cases}$$

Since the nonrenewable resource penalty is considered as a second criterion to be minimized in addition to the makespan, the problem becomes bi-objective. To solve it, we consider at each generation the whole set of parents and children distributed regarding to the makespan and the penalty value criteria. It is crucial that the fitness function considers both objectives, but also avoids premature convergence. In other words, it should promote diversity in the population set. By diversity, we mean that the solutions should be dispersed all over the considered search space. Since the algorithm tends to converge into the non-dominated solutions, the algorithm should also space the range of non-dominated solutions. A solution s is non-dominated if there does not exist another solution s' which is better than solution s w.r.t. to at least one criterion while not being inferior w.r.t. to the remaining criteria.

To compute the fitness value, we firstly adopt the rank-based fitness assignment method for multi-objective genetic algorithms (MOGAs) conceived by Fonseca and Fleming (1993); secondly, we propose different density computation methods borrowed from well-known clustering heuristic procedures. A tutorial for evolutionary algorithms used in multi-objective context may be consulted in Coello Coello et al. (2002).

3.6.1. Rank-based fitness assignment method

The method consists in assigning a rank value to each individual depending on its position within the population, and considering the criteria to be optimized. The smaller the number of solutions dominating an individual, the smaller the rank of this individual Suppose that, at a given generation t, an individual I is dominated by a number $p_{(t)}$ of individuals in the considered population. Its rank will be determined as follows:

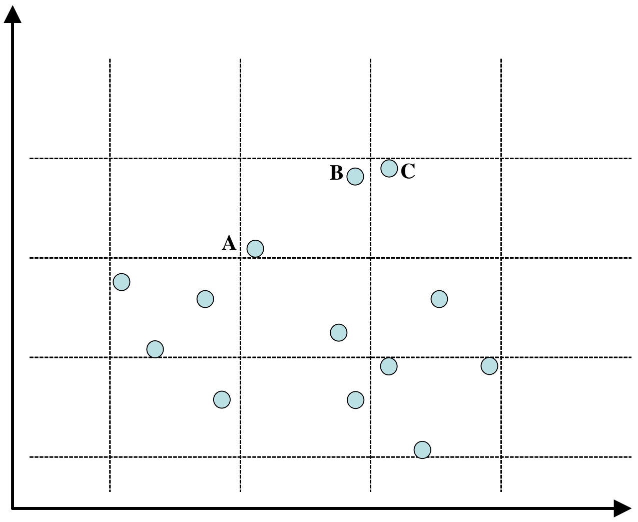

Fig. 2. Classic cell-based density approach.

$$\text{Rank}(I, t) = 1 + p_{(t)}.$$

Remark that all non-dominated solutions have a rank equal to 1. Note also that, for a given individual, this metric may varies through generations because of the population distribution changes.

3.6.2. Clustering heuristics for density computation

To avoid a premature convergence of the algorithm, we introduce density evaluation into the fitness computation. The metric will penalize individuals grouping within the population. Classic cell-based density approaches divide the objective space into cells (cf. Fonseca and Fleming, 1993; Lu and Yen, 2003) in a priori manner. Fig. 2 shows a simplified example of a population divided into cells. Although the approach is simple and quick, it presents some shortcomings. Let us consider individuals A, B and C. On the one hand, solutions A and B are regarded as belonging to the same cell whereas they are far one from the other. On the other hand, solutions B and C are very close but they belong to different cell.

Our idea is to build the cells directly on the individuals, observing their relative closeness; the cells should adapt automatically on the set of solutions. Therefore, to identify the potential groups of neighbor solutions, we propose using K-means or hierarchical clustering methods; for more details on clustering methods we refer the reader to Fung (2001). Intuitively, we expect that the new approach will overcome the limits of the above-mentioned one. Fig. 3 depicts a possible clustering result that may be obtained for the same set of solutions than in Fig. 2.

3.6.2.1. K-means algorithm. The algorithm was first conceived by MacQueen (1967). A fixed number of clusters, say K, has to be identified. In our context, K can be chosen as the worst rank value (the biggest), the number of different ranks or, a linear function of one of these measures. The K-means algorithm comes down to the next steps:

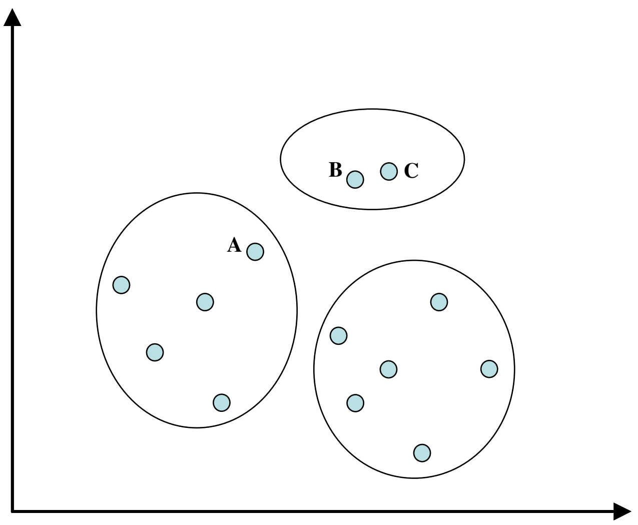

Fig. 3. Clustering-based density approach.

Step 1: From the population, choose randomly K points that represent the clusters "centroids".
Step 2: Assign individuals to the cluster that have the closest centroid.
Step 3: After assigning all individuals, recalculate the positions of the new centroids by computing, for each criterion, the average value of individuals within the same cluster.
Step 4: repeat Steps 2 and 3 until the centroids become steady or after performing a maximum number of iterations.

In our algorithm, if within iterations a centroid will not be assigned a close individual, their coordinates will be changed to (0, 0). This fact aims to regroup efficient and best individuals together and to separate them from other ones; indeed, the best individuals are those that have the least penalty and makespan. Note that since some centroids may have not been assigned close solutions, the total number of clusters can be less than the a priori determined number K. The latter is therefore an upper bound of the number of clusters. Finally, the density D of an individual amounts to the total number of individuals within its cluster.

3.6.2.2. Hierarchical method. In the hierarchical clustering, a limit distance, say d_{limit}, or a number of clusters K is fixed. d_{limit} can be the mean value or the median of the distance matrix; it can be also another function that we define later. K can be one of the values quoted in the previous subsection.

The hierarchical clustering algorithm is performed as follows:

Step 1: Identify the minimum value within the matrix distance, and join the two corresponding individuals into the same cluster.
Step 2: Compute the centroid of the new cluster.
Step 3: Update the distance matrix by introducing the new centroid considered as a new individual and removing the two old ones.
Step 4: Repeat Steps 1–3 until the minimum distance between clusters becomes greater or equal to a predefined limiting distance, that we call d_{limit}, or until the number of obtained clusters reaches K.

Several varieties of hierarchical clustering are possible; namely single-link, complete-link and average-link clustering. In average-link clustering, the coordinates of the centroid are the mean (median) of all points belonging to the same cluster. The two clusters whose two centroids have the smallest relative distance are merged. In single-link (complete-link) clustering, the two clusters whose two closest (distant) members have the smallest distance are merged. That is, "the similarity between two clusters is the similarity of their most similar (dissimilar) members". In this case, we can imagine fictitious centroids whose coordinates have the minimum (maximum) distance from other clusters.

Similarly, the density D of an individual is the total number of individuals within its cluster. In both algorithms, we apply the Euclidian and the Manhattan distance.

3.6.3. Fitness function

We compute the fitness value of an individual I in a population at a generation t by:

$f(I,t) = \dfrac{1}{\text{rank}(I,t) \times D(I,t)}$ when the density is considered and,
$f(I,t) = \dfrac{1}{\text{rank}(I,t)}$ when the density is ignored, which comes to fix the density to 1 for all individuals within generations.

Note that the fittest individual that may occur is a non-dominated solution which is alone within its cluster. Consequently, this individual would have a rank and a density equal to one. Therefore, the fitness value will be, in its turn, equal to one. Other individuals would have greater rank and/or a higher density and would then be assigned a fitness value less than one.

3.7. Crossover

In this phase, we randomly select two individuals from the current population. Individuals belonging to the current population should not be selected twice within one generation. For our algorithm, we choose the one point crossover for both components of the chromosome: the activity list and the mode assignment.

Let us consider the adapted one point crossover approach used by Hartmann (2001). We select two parents $I^{\mathcal{M}}$ and $I^{\mathcal{F}}$ embodied as follow:

$$I^{\mathcal{M}} = (j_1^{\mathcal{M}}, \ldots, j_J^{\mathcal{M}}) \text{ and } I^{\mathcal{F}} = (j_1^{\mathcal{F}}, \ldots, j_J^{\mathcal{F}}).$$

A random number q_{cross1} is generated randomly, such as $1 \leqslant q_{\text{cross1}} < J$. A daughter $I^{\mathcal{D}}$ and a son $I^{\mathcal{S}}$ are produced. $I^{\mathcal{D}}$ is defined as follows:

The q_{cross1} first jobs are taken from the mother; so, for the positions $i = 1, \ldots, q_{\text{cross1}}$ we set: $j_i^{\mathcal{D}} := j_i^{\mathcal{M}}$.

For the remaining positions (i.e. $i = q_{\text{cross1}} + 1, \ldots, J$), activities are taken from the father from the set of activities not already scheduled. That is $j_i^{\mathcal{D}} := j_k^{\mathcal{F}}$, where k is the lowest index such that $j_k^{\mathcal{F}} \notin \{j_h^{\mathcal{D}} : h = 1, \ldots, i-1\}$.

Fig. 4 illustrates the operation by an example; set $q_{\text{cross1}} = 3$. The first three activities of the daughter are taken from the mother; the remaining is taken from the father: we begin by the first job (1); it is already scheduled. The second job (3) is not yet scheduled, so we put it, etc. We proceed similarly for the son by replacing the mother by the father and the father by the mother.

After applying the operation on the activity list, q_{cross2} is drown randomly such that $1 \leqslant q_{\text{cross2}} < J$; suppose $q_{\text{cross2}} = 4$. Fig. 5 above illustrates the process.

The first q_{cross2} modes in the daughter are taken from the mother; remaining modes are taken from the father. Similarly, the son takes the q_{cross2} first modes from the father and the

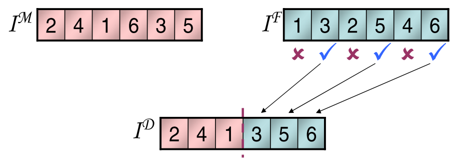

Fig. 4. One-point crossover for activity list representation; producing the daughter; $q_{\text{cross1}} = 3$.

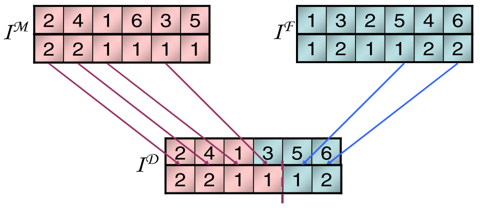

Fig. 5. One-point crossover applied on mode assignments, producing the daughter mode. $q_{\text{cross2}} = 4$.

remaining from the mother. More explicitly, modes of activities in the daughter I^D are defined as follows:

$$m^D(j_i^{\mathcal{D}}) := \; m^{\mathcal{M}}(j_i^{\mathcal{D}}) \; i = 1, \ldots, q_{\text{cross2}};$$
$$m^{\mathcal{D}}(j_i^{\mathcal{D}}) := \; m^F(j_i^{\mathcal{D}}) \; i = q_{\text{cross2}} + 1, \ldots, J.$$

In the example above, the first 4 activities in daughter $I^{\mathcal{D}}$ are defined from the mother, even if the fourth job (job 3) is taken from the father. The fifth and sixth jobs have their modes from the father.

3.8. Mutation

The mutation operator is applied on newly generated individuals with a probability of mutation p_{mut}. This operator is applied, first, on activity list string and, second, on the mode assignment one. In the first sub-stage, we randomly choose a position q_{mut} such that $1 \leqslant q_{\text{mut}} < J$; then we check whether jobs in the positions q_{mut} and $q_{\text{mut}} + 1$ can be permuted; that is if the job in position $q_{\text{mut}} + 1$ is not an immediate successor of the job in position q_{mut}, we can permute the jobs; otherwise, we choose another position q'_{mut}. We repeat the procedure until two jobs are permuted or until J unsuccessful attempts are made. Note that at this stage, the mode assignment is not affected; that is permuted jobs keep their initially assigned modes. Afterwards, we randomly select one job j which has more than one mode alternative. From the set of modes $\mathcal{M}_j$, we randomly assign a mode $m'(j)$ different from the current one $m(j)$. The proposed mutation operator is different from those of Hartmann (2001) and Alcaraz et al. (2003). In fact, this mutation leads to an individual very close to the initial (in his neighborhood); whereas the mutations proposed by the latter authors carry out to completely different individuals. Consequently, we expect to need a higher mutation probability than those authors.

3.9. Selection

In this step, we apply the ranking and the roulette wheel methods. The processes consist in considering both the current and the newly generated populations. Then, *POP* individuals are selected. Note that with roulette wheel method, an individual can be chosen more than once.

Table 1
Average deviations from optimal solutions. EA with *K*-means clustering stopped at a number of clusters equal to the number of different ranks. J20; 1 s stopping criterion.

p_{mut}	*POP*			
	30 (%)	60 (%)	90 (%)	120 (%)
0.1	–	1.89	1.60	1.60
0.5	1.71	1.27	1.18	1.52
0.9	1.37	1.02	1.07	1.71

4. Computational results

4.1. Test problems

The experiments have been performed on a Pentium4-based IBM-compatible personal computer with 3.00 GHz clock-pulse and 512 Mb RAM. The EA has been compiled in with Microsoft Visual C++ 6.0 compiler and tested under Windows XP professional. We ran our EA program on standard MRCPSP test instances. These problems as well as their best found solutions are available in the project scheduling problem library PSPLIB; for more details, we refer the reader to Kolisch and Sprecher (1996). Since the majority of works in the literature treat only J10-20 and J30, we choose to follow them in order to be able to carry out the comparisons. Optimal solutions for J10-20 instances sets are available, whereas no optimal solutions were found for J30 instances; hence, heuristic lower bounds are provided for the latter.

4.2. Configuration of the algorithm

In this section, we report the numerical configuration of our algorithm through a numerical investigation.

Following Hartmann (2001), the performances of the algorithm have been investigated on the set J20 since it is the hardest standard set for MRCPSP whose all optimal solutions are known; besides, according to Hartmann (2001), "the results for the smaller projects are similar". The experiments have been conducted with 1 s running time stopping criterion. The best configuration results consist in the following issues: the preprocessing and the local search improvement procedures deserve to be applied; the best GA parameters are a population with 60 individuals and a mutation probability equal to 0.90; the Manhattan distance is chosen; the clustering step is efficient, provided that the stopping criterion is the number of computed schedules; otherwise, the rank-based fitness is more suitable.

4.2.1. Population size and mutation probability

On the one hand, a high mutation probability implies that the operator is frequently used, and this may induce to a high time consumption over generations; on the other hand, it is evident that a high population size requires more computation effort than a low one. Besides, the number of performed generations depends on the computational effort and on the given time limit. In fact, great population size and mutation probability require a long computation time; so, for a limited CPU time, the algorithm computes entirely a low number of schedules. Consequently, we choose to experiment both population size and mutation probability on the basis of 1 s stopping criterion. The average deviations from the optima are reported in Table 1. The investigation on population size and mutation probability leads to best results with *POP* = 60 and $p_{\text{mut}} = 90\%$.

The algorithm has also been tested with Hartmann's GA mutation and the best result has been found with a low mutation probability. Hartmann's operator leads to a solution wholly different from the initial (far from its neighborhood); but best results derive

Table 2
Average deviations. EA with fitness function based on hierarchical clustering. J20. 1 s.

(a) *POP*	30 (%)	60 (%)	90 (%)	120 (%)
Complete-link hierarchical clustering stopped at $d_{\min}$	1.46	1.22	0.92	0.88
(b) Hierarchical clustering type	Single-link (%)	Average-link (%)	Complete-link (%)	
Hierarchical clustering stopped at $d_{\min}$. *POP* 120	0.92	0.88	0.88	
Hierarchical clustering stopped at $d_{\min}$. *POP* 60	1.27	1.25	1.22	
(c) Hierarchical clustering stopping criterion	$d_{\min}$ (%)	Number of different ranks (%)		
Hierarchical clustering stopped at $d_{\min}$. *POP* 120	0.88	0.94		

Table 3
Average deviations. EA with *K*-means based fitness function. J20. 1 s stopping criterion.

K-means stopping criterion	Number of different ranks	Maximum rank
POP 60; p_{mut}=0.90	1.02%	1.14%

still from our simple mutation with high probability. The unusual high probability of mutation with our operator may be due to the nature of the selected mutation variant. In fact, the latter leads to taking another individual, but not strongly different from the first. The mutated solution belongs to the neighborhood of the initial one. Therefore, to enjoy diversification, our mutation probability has to be higher than usual.

4.2.2. Impact of the clustering alternatives

The clustering techniques used in computing the density have been explored with different stopping criteria. We studied for instance, for the hierarchical heuristic, a limiting distance which is function of the average distance or the median distance. These measures are deduced from the distance matrix between the individuals in the union of the whole population and its children. Two distance metrics are investigated: the Euclidian and the Manhattan distances. The stopping criteria may also consist in limiting the number of clusters; for example, we tested this limit to be the number of different ranks in the population and its offspring, as well as different functions of the maximum rank value. Besides, in the hierarchical clustering, single, complete and average linkages clustering have been tested. We also proposed a new limiting distance, we call $d_{\min}$, to apply to the hierarchical heuristic

$$d_{\min} = \frac{1}{2}\min\left(\frac{(T - mpm)}{POP}, \frac{L_{\max}}{POP}\right)$$

where mpm is the project makespan regarding only the precedence constraints; T is an upper bound for the project makespan. We choose T as the sum of the maximum processing times of the jobs; formally: $T = \sum_{j=1}^{J}\left(\max_{m\in\mathcal{M}_j} p_{jm}\right)$; $L_{\max}$ is the maximum penalty which could exist; namely: $L_{\max} = \max_{\substack{I\in\mathcal{POP}\cup\mathcal{CHI}\\ I=(\lambda,\mu)}} L(\mu)$.

For each individual $I = (\lambda, \mu)$, the mode vector μ leads to a penalty $L(\mu)$ computed by aggregating $L_k(\mu)$ as explained in Section 3.6. $L_{\max}$ can be zero if all nonrenewable resources are abundant; that is to say, there are no more nonrenewable resource constraints thanks to the preprocessing procedure. In this case, the hierarchical procedure is performed for only a unique iteration.

The experiments show that in all experimented cases, the use of Manhattan distance provides better results than the Euclidian. Table 2(a) presents a comparison between results obtained when the population size is varied. Note that, although the high mutation probability aims to keep the diversification within the EA, the least average deviation is get with a population size equal to 120.

For a given time limit, high population size uses to maintain the diversification within the population instead of advancing longer along generations. Then, the hierarchical clustering types, namely single, average and complete-link, have been evaluated. The derived results are reproduced in Table 2(b). Complete-link hierarchical clustering leads to slightly better results than single and average-link ones. Since different stopping criteria for the hierarchical clustering are proposed, we present the results of the most promising ones in Table 2(c); namely, $d_{\min}$ and the number of different ranks. The best result derives from the hierarchical clustering stopped when the minimum distance between two clusters becomes equal to $d_{\min}$.

Moreover, Table 3 shows that the *K*-means gives better results when it is stopped with a number of clusters equal to the number of different ranks, rather than when it is stopped with a number of clusters equal to the maximum rank. Indeed, by observing the number of performed generations in both cases, it appears that the algorithm performing with the number of different ranks reaches on average 490 generations, while the algorithm which stops the clusters quantity at the number of maximum rank performs only 224 generations. In fact, it can be deduced that the maximum rank stopping criterion requires a higher number of iterations of the clustering algorithm, without being much more effective in terms of average deviation.

Finally, in Table 4, the simple rank-based fitness function, as well as the best configurations of the hierarchical and the *K*-means clustering based fitness functions are compared. Here, the J20 set with two stopping criteria for the EA are considered. In the first line, we report the average deviations found with the algorithm tested on J20 and stopped at 1 s running time. Whereas in the second line, the deviations found with the algorithm tested on J20 and stopped at 5000 computed schedules are displayed. In fact, we choose to experiment the EA with this set and this stopping criterion since they will be used as a basis of comparison with other algorithms. It appears that the deductions are different according to the stopping criterion chosen. Obviously, with 1 s stopping criterion, the best average deviation is obtained with a simple rank-based fitness function. Whereas, with 5000 computed schedule criterion, the best average deviation is found with the *K*-means clustering stopped at the number of different ranks when the population size is fixed to 60. The simple rank and the hierarchical clustering-based fitness functions provide more or less the same results.

Noticeably, the simple rank-based fitness function requires less time than the others; hence, for a determined time limit, it gives the best results. However, when the stopping criterion is the number of computed schedules, the *K*-means clustering with a population of 60 is preferred. In other words, when time is available, clustering can be useful to the optimization process.

The comparison between the three fitness variants is made on the basis of average deviations with respect to optimal values, and the percentage of instances to which an optimal solution is found (cf. Table 5).

Table 4
Comparison between the different fitness function computations. *POP* = 120, $p_{\mathrm{mut}} = 0.90$.

		Rank-based fitness function ($POP = 120, p_{\mathrm{mut}} = 0.90$)	Hierarchical clustering stopped at $d_{\min}$ ($POP = 120, p_{\mathrm{mut}} = 0.90$)	*K*-means clustering stopped at the number of different ranks ($POP = 60, p_{\mathrm{mut}} = 0.90$)	*K*-means clustering stopped at the number of different ranks ($POP = 90, p_{\mathrm{mut}} = 0.90$)
J20, 1 s Av. dev.		0.75%	0.88%	1.02%	1.07%
J20, 5000 computed schedules	Av. dev.	2.44%	2.40%	1.62%	2.09%
	CPU	0.124 s	0.308 s	0.263 s	0.429 s

Table 5
Average deviations and optimal solutions 5000 computed schedules.

	J10	J12	J14	J16	J18	J20
(a) Av. Dev (%)						
New EA with simple rank-based fitness function (*POP* = 120, GEN = 41)	0.17	0.29	0.94	1.23	1.78	2.44
K-means based EA with different ranks clusters number (*POP* = 60, GEN = 83)	0.21	0.29	0.77	0.91	1.30	1.62
K-means based EA with different ranks clusters number (*POP* = 90, GEN = 55)	0.14	0.24	0.80	1.14	1.53	2.09
Hierarchical clustering-based EA with $d_{\min}$ limiting distance (*POP* = 120, GEN = 41)	0.16	0.27	0.90	1.22	1.71	2.43
(b) Optimal (%)						
New EA with simple rank-based fitness function (*POP* = 120, GEN = 41)	96.75	94.14	81.67	75.27	66.73	59.87
K-means based EA with different ranks clusters number (*POP* = 60, GEN = 83)	96.19	94.2	83.84	80.69	72.92	67.43
K-means based EA with different ranks clusters number (*POP* = 90, GEN = 55)	97.31	94.94	83.18	77.27	70.78	63.16
Hierarchical clustering-based EA with $d_{\min}$ limiting distance (*POP* = 120, GEN = 41)	97.01	93.90	81.53	75.76	68.22	60.01

Table 6
Comparison with other heuristics. Our fitness including *K*-means with different ranks clusters number.

(a) Set J10, 6000 schedules	Av. dev. (%)	Optimal (%)
Hartmann (2001)	**0.10**	**98.1**
K-means based EA with different ranks clusters number (*POP* = 90, GEN = 66)	**0.11**	**97.8**
Hierarchical clustering-based EA with $d_{\min}$ (*POP* = 120, GEN = 50)	0.15	97.31
Rank-based EA (*POP* = 120, GEN = 50)	0.16	97.14
Alcaraz et al. (2003)	0.19	96.5
Kolisch and Drexl (1997)	0.50	91.8
Özdamar (1999)	0.86	88.1

(b) Set J10, adjusted CPU time.		*POP* = 30	*POP* = 60	*POP* = 90	*POP* = 120
Our EA with Hartmann's corrected fitness function	Av. dev. (%)	0.81	0.37	0.31	0.30
	Optimal (%)	88.87	93.99	94.96	95.1
	Schedules	6000	6000	6000	6000
Our EA with simple rank-based fitness function	Av. dev. (%)	0.47	**0.25**	**0.26**	0.36
	Optimal (%)	92.68	**95.59**	**95.67**	93.65
	Schedules	5331	4620	3879	3408
Our EA with hierarchical clustering	Av. dev. (%)	0.79	0.58	1.38	3.31
	Optimal (%)	88.58	90.15	80.97	61.45
	Schedules	2229	1806	1341	1032
Our EA with *K*-means	Av. dev. (%)	0.80	2.63	4.82	6.49
	Optimal (%)	88.58	70.19	54.96	37.61
	Schedules	2280	1446	1260	1020

(c) Av. dev (%), 5000 schedules	J10	J12	J14	J16	J18	J20
K-means based EA with different ranks clusters number (*POP* = 60, GEN = 83)	0.21	0.29	**0.77**	**0.91**	1.30	**1.62**
K-means based EA with different ranks clusters number (*POP* = 90, GEN = 55)	**0.14**	**0.24**	0.80	1.14	1.53	2.09
Ranjbar et al. (2009)	0.18	0.65	0.89	0.95	**1.21**	1.64
Alcaraz et al. (2003)	0.24	0.73	1.00	1.12	1.43	1.91
Józefowska et al. (2001)	1.16	1.73	2.6	4.07	5.52	6.74

In all tested sets, the hierarchical clustering leads to similar results, even slightly better than those of the simple rank-based fitness function. However, with the stopping criterion of the number of computed schedules, we suppose that the two methods require the same computational effort to construct schedules; this is not the case since the hierarchical clustering is added the simple rank-based fitness function to obtain the hierarchical clustering-based fitness function. The investigation of the computational time reveals that with hierarchical clustering, the algorithm lasts more than 2.3 times than without clustering (the difference between elapsed times depend on the set explored). The *K*-means based EA requires more computational effort than the simple one, but it still requires less computational time than the hierarchical. Moreover, the method gives the best results in terms of average deviations and percentage of instances to which an optimal solution is found. Although, for the sets J10 and 12, the 60 population-sized EA gives worse results than the 90 population-sized one, we prefer it because it outperforms the second and all other EA proposed variants for the set J14; furthermore, the differences between deviations for the sets J10 and J12 are not very large.

Table 7
Comparison with simulated annealing algorithm EA with *K*-means based fitness function stopped at number of different ranks.

J	Av. dev.		Max. Dev.		Optimal	
	Our EA[a]	BLSA[b]	Our EA[a]	BLSA[b]	Our EA[a]	BLSA[b]
10	0.08	0.21	8.80	7.8	98.5	96.3
12	0.12	0.19	7.10	6.3	97.5	91.2
14	0.42	0.92	12.25	10.6	90.6	82.6
16	0.60	1.43	13.68	12.9	85.9	72.8
18	0.71	1.85	11.11	11.7	83.1	69.4
20	1.02	2.10	14.09	13.2	76.7	66.9

[a] Pentium 3.00 GHz, time limit 1 s.
[b] Pentium 100 MHz, time limit five times the instance size (in seconds).

4.3. Comparison with other algorithms

In the literature of MRCPSP, comparisons are usually made on the basis of two stopping criteria types: the CPU time and the number of generated schedules. The first supposes that all algorithms are tested using similar computer frameworks, the same programming language and the same ability in translation into source code. This is of course awkward. Nevertheless, the comparisons have been conducted while indicating the computer framework used.

Analogously, the number of computed schedules criterion presumes that schedules require the same computational effort in all algorithms; this hypothesis should also be revised because the computational effort required to generate schedules is strongly conditioned by the operator types used and the heuristics possibly introduced. For instance, it is clear that the clustering heuristics we introduced require additional computational effort. To overcome this drawback, a comparison is made while the CPU time stopping criterion is adjusted in order to make the experiments comparable.

Hence, in the remaining, in order to be able to compare our algorithm with those proposed in the literature of the problem, we are constrained to resort to the same bases of comparison. So, results will be displayed using these three stopping criteria: the number of computed schedules, the CPU time and the adjusted CPU time (as explained below).

In this subsection, we compare the variants leading to the best configurations on the different set instances with 5000 computed schedules stopping criteria with other algorithms, namely Alcaraz et al. (2003) and Józefowska et al. (2001). We also compare our EA with the GAs of Alcaraz et al. (2003), Hartmann (2001) and Özdamar (1999), and the local search algorithm of Kolisch and Drexl (1997) on the basis of 6000 computed schedules stopping criterion. Furthermore, we experiment the algorithm while applying the corrected Hartmann's fitness; the algorithm is stopped after computing 6000 schedules; afterwards, the average spent CPU time is used as a stopping criterion to our algorithm with its different versions. Such an adjusted CPU time criterion allows a fair comparison between Hartmann's algorithm and our proposal. Indeed, it implicitly decreases the number of schedules generated by the bi-objective approach according to the increase of computation time needed to calculate the rank and the density. The experiments are conducted on J10 set (see Table 6).

Hartmann (2001) has conceived a GA which is, in our knowledge, the best algorithm proposed in the literature of MRCPSP. Hartmann's GA (HGA) leads to the best results when the stopping criterion is 6000 schedules (J10), even though they are close to ours.

In Table 6(b), the average deviations, percentage of optimal solutions and number of performed schedules are displayed, when the adjusted CPU time criterion is used; i.e. the algorithm is stopped at the average time spent by the corrected Hartmann's fitness algorithm to compute 6000 schedules. The different algorithm variants are tested cross different population sizes. It appears that, even though the number of performed generations is reduced, our algorithm with simple rank fitness function outperforms other variants; i.e. it is better to spend additional computational effort to compute simple rank fitness rather than to go more along generations. It is worthwhile to note that other variants are of interest for large-sized instances as shown earlier in Table 5.

Our EA is also compared with Ranjbar et al. (2009), Alcaraz et al. (2003) and Józefowska et al. (2001) for 5000 computed schedules stopping criterion. The investigation is made on J10-20 and J30 sets. The results are displayed in Table 6(c) above.

Table 8
Comparison with HGA and SDBB. EA with *K*-means based fitness function stopped at number of different ranks. 1 s stopping criterion. J10-20 and J30. (Deviations are computed from makespan lower bounds for J30.)

J	Av. dev.			Max. dev.			Feasible			Optimal		
	Our EA	HGA	SDBB	Our EA	HGA	SDBB	Our EA	HGA	SDBB	Our EA	HGA	SDBB
10	0.09	0.06	**0.00**	13.04	6.3	0.0	100.0	100.0	100.0	98.6	98.7	100.0
12	0.13	0.14	**0.12**	**8.69**	9.1	17.9	100.0	100.0	100.0	97.3	97.3	**98.2**
14	**0.43**	0.44	1.46	12.13	**10.3**	33.3	100.0	100.0	99.6	**90.0**	89.8	85.7
16	**0.46**	0.59	3.81	13.79	**10.5**	52.4	100.0	100.0	99.5	**88.9**	87.8	69.5
18	**0.67**	0.99	7.48	**13.24**	13.3	77.4	100.0	100.0	98.0	**84.1**	78.3	57.4
20	**0.91**	1.21	11.51	21.53	**14.2**	78.6	100.0	100.0	96.4	**78.52**	73.3	47.3
30	**2.04**	16.93	57.22	**14.55**	151.9	244.0	86.17	86.3	55.8	–	–	–

Table 9
Comparison with JPS and DDE.; simple-rank-based EA with *POP* = 120; *K*-means based EA with *POP* = 60; 0.15 s/activity stopping criterion.

J	Av. dev.				**Optimal**			
	Simple rank EA	*K*-means EA	JPS	DDE	Simple rank EA	*K*-means EA	JPS	DDE
10	**0.03**	0.09	0.03	0.09	**99.25**	97.95	99.25	99.3
12	**0.05**	0.12	0.09	0.11	**98.72**	97.44	98.47	99.3
14	**0.26**	0.3	0.36	0.34	**93.93**	92.56	91.11	97.6
16	**0.25**	0.45	0.44	0.42	**93.09**	89.45	85.91	96.38
18	**0.45**	0.7	0.89	0.59	**88.22**	84.06	79.89	94.43
20	**0.58**	0.91	1.10	0.7	**84.81**	77.76	74.19	91.75

Alcaraz et al. (2003) conceived a GA which is inspired from Hartmann's algorithm (HGA). The main distinction appears in the new fitness function and operators variants they propose. Table 6 shows that our algorithm with a 60 sized population gives better results than their GA for all explored sets, and for 5000 schedules stopping criterion. Moreover, for 6000 computed schedules, and regarding J10 set, the three EA variants outperform Alcaraz et al.'s GA (2003).

The table classes Özdamar's (1999) GA results in the last position. Obviously, the encoding she proposed does not succeed to give good solutions for the problem. Kolisch and Drexl's (1997) local search procedure succeeds to find better results than that of Özdamar (1999), but it does not outperform ours. The simulated annealing proposed by Józefowska et al. (2001) finds average deviations that are more than three times worse than our EA.

Table 7 summarizes the average and maximum deviations from the optimums found in J10-20 sets on the basis of 1 s stopping criterion; the percentage of instances where optima are found is also evoked. The comparison is made between our EA and the Simulated Annealing of Bouleimen and Lecocq (2003): BLSA. Even if it is made on different processor frameworks, the latter's time limit is 5 times the instance size in seconds, while our algorithm's stopping criterion remains 1 s in all instance sizes. For all tested sets, and in terms of average deviation and optimal solutions, our algorithm tallies best values.

We deduce from the results in Table 8 that Sprecher and Drexl's truncated branch-and-bound (SDBB) outperforms our EA in J10 and J12 instances set since it provides entirely optimal solutions; whereas, for larger problem sizes, our EA performs much better even if our stopping criterion was 1 s. Although the truncated branch-and-bound algorithm provides optimal solutions for the whole set of J10 instances, it does not perform efficiently for J18 and J20 sets. Observe that, for the 20 activity set, the average deviation derived from Sprecher and Drexl's algorithm is more than nine times higher than the one resulting from the implementation of our EA. In addition, the average deviation they find for J30 is more than 28 times worse than the average deviation found by the proposed EA.

Our EA has also been compared with the Particle Swarm algorithm applied by Jarboui et al. (2008) and the Differential Evolution proposed by Damak et al. (2009) implemented on the same computer framework; we call them JPS and DDE respectively. Hence, the related results reported in Table 9 show clearly the outperforming of our EA versus these above-mentioned methods in all the instance sets J10-20.

5. Conclusion

In this study, we propose a new EA to solve the MRCPSP. Because of the NP-completeness of the problem of finding a feasible solution of MRCPSP, we have decided to allow nonrenewable resources violation, and a penalty value is assigned to the issued solutions. The penalty is dealt as a criterion to be minimized; hence, the algorithm solves the problem becoming bi-objective.

New methods to deal with multi-objective evolutionary algorithm (MOEA) are proposed. In fact, individuals are evaluated by the well-known ranked-based fitness assignment method; besides, we introduce clustering algorithms to compute densities. In this way, we enforce that neighbor solutions belong to the same cluster and are assigned the same density. In other words, our proposal comes down to an adaptive grid in the solution set.

Other interesting ideas were proposed in the resource-constrained project scheduling problem (RCPSP) literature such as the forward-backward improvement, some specific operators. We quote for example Alcaraz et al. (2004), Debels et al. (2006), Hartmann (2002), Kochetov and Stolyar (2003), and Valls et al. (2003, 2005). These propositions outclass classical techniques solving the RCPSP. Their adaptation to solve the MRCPSP can be promising but it is not the subject of the present paper. It can constitute the subject of later studies.

It would be interesting to apply our two main proposals to other problems since they look like promising. For problems where feasible solutions are hard to find, the first proposal consists in transforming some constraint violation into an additional criterion: this allows working with partially unfeasible solutions but also transforms single objective problems into bi-objective one. For any multi-objective evolutionary algorithm, the second proposal relies on the use of clustering techniques to compute fitness on adaptive grid density, in order to promote search diversity. The clustering step in the fitness evaluation induces additional computational effort; but the latter can be, for some optimization problems, negligible with respect to the computational effort need to evaluate the plain objective. In such cases, this clustering step requires attention.

References

Alcaraz, J., Maroto, C., Ruiz, R., 2003. Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of Operation Research Society 54, 614–626.

Alcaraz, J., Maroto, C., Ruiz, R., 2004. Improving the performance of genetic algorithms for the RCPS problem. In: Proceedings of the Ninth International Workshop on Project Management and Scheduling Nancy, pp. 40–43.

Bianco, L., Dell'Olmo, P., Speranza, M.G., 1998. Heuristics for multimode scheduling problems with dedicated resources. European Journal of Operational Research 107, 260–271.

Boctor, F.F., 1993. Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31, 2547–2558.

Boctor, F.F., 1996a. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research 90, 349–361.

Boctor, F.F., 1996b. Resource-constrained project scheduling by simulated annealing. International Journal of Production Research 34, 2335–2351.

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

Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B., 2002. Evolutionary algorithms for solving multi-objective problems. In: Book Series on Genetic Algorithms and Evolutionary Computation May. Kluwer Academic Publishers.

Damak, N., Jarboui, B., Siarry, P., Loukil, T., 2009. Differential evolution for solving multi-mode resource-constrained project scheduling problems. Computers and Operations Research 36, 2653–2659.

Debels, D., De Reyck, B., Leus, R., Vanhoucke, M., 2006. A hybrid scatter search/ electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169, 638–653.

De Reyck, B., Herroelen, W., 1999. The multi-mode resource-constrained project scheduling problem with generalized precedence relations. European Journal of Operational Research 119, 538–556.

Drexl, A., Grüenewald, J., 1993. Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 25, 74–81.

Elloumi, S., Loukil, T., Teghem, J., 2006. Ordonnancement de projets multi-mode sous contraintes de ressources: Un algorithme évolutionnaire basé sur l'information de l'efficacité. ValenSciences, vol. 5. Presses Universitaires de Valenciennes, pp. 237–252.

Elmaghraby, S.E., 1977. Activity Networks: Project Planning and Control by Network Models. Wiley, New York.

Fonseca, C.M., Fleming, P.J., 1993. Genetic algorithms for multi-objective optimization: formulation, discussion and generalization. In: Forrest, S. (Ed.), Genetic Algorithms: Proceedings of the Fifth International Conference. Morgan Kaufmann, San Mateo, CA, July.

Fung, G., 2001. A Comprehensive Overview of Basic Clustering Algorithms. <http://pages.cs.wisc.edu/~gfung/clustering.pdf>.

Hartmann, S., 2001. Project scheduling with multiple modes: a genetic algorithm. Annals of Operations Research 102, 111–135.

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

Hartmann, S., Drexl, A., 1998. Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32, 283–297.

Jarboui, B., Damak, N., Siarry, P., Rebai, A., 2008. A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation 195, 299–308.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., Weglarz, J., 2001. Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research 102, 137–155.

Kochetov, Y., Stolyar, A., 2003. Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem. In: Proceedings of the Third International Workshop of Computer Science and Information Technologies, Russia.

Kolisch, R., 1996. Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. European Journal of Operational Research 90, 320–333.

Kolisch, R., Drexl, A., 1997. Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 29, 987–999.

Kolisch, R., Sprecher, A., 1996. PSPLIB – a project scheduling problem library. European Journal of Operational Research 96, 205–216.

Lu, H., Yen, G.G., 2003. Rank-density-based multiobjective genetic algorithm and benchmark test function study. IEEE Transactions on Evolutionary Computation 7, 325–343.

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. In: Le Cam, L.M., Neyman, J. (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. 1. pp. 281–297.

Özdamar, L., 1999. A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews 29, 44–59.

Patterson, J.H., Słowiński, R., Talbot, F.B., Węglarz, J., 1989. An algorithm for a general class of precedence and resource constrained scheduling problems. In: Słowiński, R., Węglarz, J. (Eds.), Advances in Project Scheduling. Elsevier, pp. 3–28.

Patterson, J.H., Słowiński, R., Talbot, F.B., Węglarz, J., 1990. Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems. European Journal of Operational Research 79, 49–68.

Ranjbar, M., De Reyck, B., Kianfar, F., 2009. A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. European Journal of Operational Research 193, 35–48.

Słowiński, R., 1981. Multiobjective network scheduling with efficient use of renewable and nonrenewable resources. European Journal of Operational Research 7, 265–273.

Słowiński, R., Węglarz, J., 1978. Solving the general project scheduling problem with multiple constrained resources by mathematical programming. Lecture Notes in Control and Information System 7, 278–289.

Słowiński, R., Soniewicki, B., Węglarz, J., 1994. DSS for multiobjective project scheduling. European Journal of Operational Research 79, 220–229.

Sprecher, A., 1994. Resource-Constrained Project Scheduling: Exact Methods for the Multi-mode Case. Lecture Notes in Economics and Mathematical Systems, vol. 409. Springer, Berlin.

Sprecher, A., Drexl, A., 1998. Multi-mode resource-constrained project scheduling by a simple general and powerful sequencing algorithm. European Journal of Operational Research 107, 431–450.

Sprecher, A., Hartmann, S., Drexl, A., 1997. An exact algorithm for project scheduling with multiple modes. OR Spektrum 19, 195–203.

Talbot, F.B., 1982. Resource-constrained project scheduling with time-resource tradeoffs: the nonpreemptive case. Management Science 28, 1197–1210.

Valls, V., Ballestin, F., Quintanilla, M.S., 2003. A Hybrid Genetic Algorithm for the RCPSP. Technical Report, Department of Statistics and Operations Research, University of Valencia.

Valls, V., Ballestin, F., Quintanilla, M.S., 2005. Justification and RCPSP: a technique that pays. European Journal of Operational Research 165, 375–386.

Węglarz, J., 1980. Multiprocessor scheduling with memory allocation: a deterministic approach. IEEE Transactions on Computers 29, 703–709.