Computers & Operations Research 36 (2009) 2653 – 2659

Contents lists available at ScienceDirect

Computers & Operations Research

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

Differential evolution for solving multi-mode resource-constrained project scheduling problems

N. Damak[a], B. Jarboui[a], P. Siarry[b,*], T. Loukil[a]

[a]FSEGS, route de l'aéroport km 4.5, B.P. No. 1088, Sfax 3018, Tunisia
[b]LiSSi, Université de Paris 12, 61 avenue du Général de Gaulle, 94010 Créteil, France

ARTICLE INFO

Available online 3 December 2008

Keywords:
Differential evolution
Scheduling
MRCPSP
Makespan

ABSTRACT

In this paper we consider the resource-constrained project scheduling problem with multiple execution modes for each activity and minimization of the makespan. To solve this problem, we propose a differential evolution (DE) algorithm. We focus on the performance of this algorithm to solve the problem within small time per activity. Finally, we present the results of our thorough computational study. Results obtained on six classes of test problems and comparison with other algorithms from the literature show that our algorithm gives better solutions.

1. Introduction

In the resource-constrained project scheduling problem (RCPSP), non-preemptive activities requiring renewable resources and subject to precedence constraints have to be scheduled to minimize the makespan. The activities have to be not interrupted during their processing. The resource constraints refer to limited renewable resources, such as manpower, material and machines, which are necessary for carrying out the project activities. The precedence constraints guarantee a logical process of the project activities.

This problem has been extended to the multi-mode resource-constrained project scheduling problem (MRCPSP), which is more related to the real world, where each activity is executed in one of several modes, which represents a combination between its resources requirements and its duration.

The MRCPSP is a *NP*-hard problem. It is an important and challenging problem that has received increasing attention for several years. Various techniques have been proposed to solve this problem.

Nevertheless, Sprecher and Drexl [1] have shown that exact methods cannot solve problems with more than 20 tasks executed on three modes and highly resource-constrained. Accordingly, meta-heuristics are being successfully applied to this problem. Hence, in practice, heuristic algorithms to generate near-optimal schedules for larger projects are of special interest. The most recent heuristic procedures of the problem that can be found are the following ones: Drexl and Grünewald [2] suggest a regret-based biased random sampling approach, Slowinski et al. [3] describe a single-pass approach, a multi-pass approach, and a simulated annealing algorithm and Kolisch and Drexl [4] present a local search procedure.

In 1999, Özdamar [5] proposed a genetic algorithm based on a priority rule encoding. Bouleimen and Lecocq [6] suggested a simulated annealing heuristic. Sprecher and Drexl [1] developed a branch-and-bound procedure which is, according to the results obtained by Hartmann and Drexl [7], the currently most powerful algorithm for solving exactly the MRCPSP. Sprecher and Drexl [1] suggested using it as a heuristic by imposing a time limit. Very recently, in 2007, Jarboui et al. [8] investigated the particle swarm optimization to solve this challenging problem. Boctor [9,10] presented heuristics for multi-mode problems without non-renewable resources.

The differential evolution (DE) is a direct search method based on population generation. This method proves itself by its extensive application areas, its simplicity and its robustness, that have promoted its reputation in many fields. It was proposed by Storn and Price in 1999 [11]. In the same year, DE was applied to problems involving multiple criteria, as a spreadsheet solver application [12]. New areas of interest also emerged, such as heat transfer [13], information fusion [14], gear design [15], and constraint satisfaction problems [16], to name only a few. In 2000, the popularity of DE continued to grow in areas of electrical power distribution [17], pharmacology [18], aeronautics [19], magnetics [20], and chemical engineering [21]. In 2001 there were further extensions of DE in areas of environmental science [22], linear system models [23], swarm technology [24], and geology [25]. By the year 2002, DE penetrated the field of medical science [26]. More recently, in 2003, there has been a resurgence of interest in applying DE to problems involving multiple

* Corresponding author.
E-mail address: siarry@univ-paris12.fr (P. Siarry).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.11.010

criteria [27,28]. In 2006, Lorenzoni et al. [29] designed an algorithm based on an extension of the DE algorithm to solve a MRCPSP.

In this paper, we propose to use the DE to solve a large variety of combinatorial optimization problems, namely the MRCPSP using integer values. The remainder of the paper is organized as follows: Section 2 describes the MRCPSP. Section 3 defines the DE algorithm. In Section 4 we adapt DE algorithm to the MRCPSP. Experimental results are reported in Section 5. Finally, Section 6 concludes the paper.

2. Problem description

The MRCPSP may be stated as follows: a project consists of a set of activities J where each activity has to be processed without interruption. The dummy activities 1 and n represent the beginning and the end of the project. Each activity j, $j = 1 \dots J$ has to be executed in one of M_j modes, where each activity-mode combination has a fixed duration. Each mode requires a constant amount of one or more of R types of renewable and non-renewable resources for the entire activity duration. An activity j started in mode $m \in 1 \dots M_j$ must be completed in mode m without preemption. The duration of activity j executed in mode m is d_{jm}. There are K renewable resource types. The availability of each resource type k in each time period is R_k units, $k = 1 \dots K$ and the number of available units of non-renewable resource l, $l = 1 \dots N$, is N_l. Each activity j requires r_{jmk} units of resource k during each period of its duration executed with mode m and each activity j requires n_{jml} units of resource l during each period of its duration executed with mode m. All parameters are assumed to have non-negative integer values. Activities are subject to finish–start precedence relations with zero time lags, meaning that an activity can be started if and only if all its predecessors have been completed. Minimization of project duration is generally considered as the objective for the MRCPSP. In addition, the activities in progress are not allowed to be interrupted. Therefore, the goal of solving the MRCPSP is to find a mode combination for all activities, as well as the resultant schedule (including starting times and resource allocation policies for all activities) that leads to minimal project duration.

3. DE method

The domain of the evolutionary algorithms has known a large development these last years. DE is one of these algorithms. At the origin, DE was conceived for continuous optimization problems without constraints. Its present extensions can handle problems of mixed variables and can manage non-linear constraints. Currently, an important number of industrial and scientific applications make use of DE. One can classify DE among the stochastic optimization metaheuristics. According to an accepted classification, DE is inspired by the genetic algorithms and the evolutionary strategies, combined with a geometric search technique. The genetic algorithm changes the structure of the individuals, while using the mutation and the crossover, whereas the evolutionary strategy achieves the auto-adaptation, through a geometric manipulation of the individuals. These ideas have been set in motion thanks to an operation, simple and yet powerful, of mutation of vectors, proposed in 1995 by Storn and Price [30].

Since then, DE became an inescapable method for numerous real problems or benchmarks. In the last 10 years, one can find a great deal of scientific and industrial problems solved by DE. Among them, one can mention registration and treatment of picture, optimal control multi-modal problems, optimization of chemical processes, multi-criteria decision, learning of neural networks, fitting of fuzzy functions, design in aerodynamics, polynomial approximation.

3.1. DE algorithm

DE, invented by Storn and Price in 1995 [30], is a simple yet powerful heuristic method for solving non-linear, non-differentiable and multi-modal optimization problems. This technique combines simple arithmetic operators with the classical crossover, mutation and selection operators. The key idea behind DE is a scheme for generating trial parameter vectors. Mutation and crossover are used to generate new vectors (trial vectors), and selection then determines which of the vectors will survive the next generation.

A set of D optimization parameters is called an individual, which is represented by a D-dimensional parameter vector. A population consists of NP parameter vectors X_i, G ($i = 1,2,\dots,NP$ for each generation G). According to Storn and Price, DE basic strategy can be described as follows.

3.1.1. Population structure

From the beginning, we randomly generate C_i^g vectors to create the initial population, which will be modified throughout the execution of the algorithm. Indeed, at each generation, we handle a current population, symbolized by P_C, that has already been found to be acceptable either as initial points, or by comparison with other vectors:

$$P_{C,i}^g = (C_i^g), \text{ with } i = 0, 1, \dots, N_p - 1 \text{ and } g = \text{index of current generation}$$
$$C_i^g = (c_{j,i}^g), \quad j = 0, 1, \dots, D - 1 \tag{1}$$

The indexes g and i indicate the generation and the population to which belongs a vector, respectively. $i \in [0, N_p - 1]$, N_p is the population size. In addition, j denotes the index of each element in the solution vector which contains D elements.

The main characteristic of that evolutionary algorithm, namely DE, is the exploration of the feasible space. This is mainly assured by a clever and suitable choice of the population size N_p. In other words, it should not be too small in order to avoid stagnation and to provide sufficient exploration. The increase of N_p induces the increase of the number of function evaluations; that is, it negatively affects the algorithm convergence speed.

3.1.2. Mutation

Similarly to a genetic algorithm, DE starts from two parents C_0^g and $(C_1^g - C_2^g)$ to create a child M_i^g. We shall then randomly sample three vectors to recombine them. Eq. (2) shows how to combine three different vectors to create a mutant vector M_i^g:

$$M_{j,i}^g = C_{j,0}^g + A * rand_{j,i}^g (C_{j,1}^g - C_{j,2}^g) \tag{2}$$

The scale factor A is a chosen positive number which controls the evolution rate of the population. While there is no upper limit on A, effective values seldom are greater than "1" and $rand_{j,i}^g$ is a random number between 0 and 1.

3.1.3. Crossover

Once created, the mutant vector is crossed by DE with a vector from the current population. As a result, a trial vector is obtained by applying the following procedure:

$$T_{j,i}^g = \begin{cases} M_{j,i}^g & \text{if } (r_{j,i}^g \leqslant Cr \text{ or } j = j_r) \\ C_{j,i}^g & \text{otherwise} \end{cases} \tag{3}$$

The crossover of different elements of vectors is performed, based on the value of the crossover factor $Cr \in [0,1]$. By comparison between this factor and the output of a uniform random number generator, $r_{j,i}^g$, we determine the source of each element of the trial vector. If the random number is less than or equal to Cr, the trial parameter is inherited from the mutant M_i^g; otherwise, the parameter is copied from the current vector.

3.1.4. Selection

In the selection step of the algorithm, the trial vector T_i^g is compared with the corresponding target vector C_i^g based on their objective function values. The fittest of the two individuals takes place in the next generation (Eq. (4)).

$$C_i^{g+1} = \begin{cases} M_i^g & \text{if } f(M_i^g) \leqslant f(C_i^g) \\ C_i^g & \text{otherwise.} \end{cases} \quad (4)$$

Once the current population is updated, it evolves again through mutation, crossover and selection until the optimum is located, or a predefined termination criterion is reached.

4. Adaptation of DE to the MRCPSP

As we can see, the MRCPSP consists of two different sub-problems: the assignment of modes to tasks and then the scheduling of these tasks in order to minimize the makespan of the project. In this section, we take advantage of DE to solve the MRCPSP. Clearly, a solution represents an assignment of J tasks to m modes. From the beginning, we generate all feasible sequences.

To illustrate the concepts of adapting DE to our problem, we will use the following example of assigning six tasks to two modes. Set six tasks to be scheduled under constraints of one renewable resource and another non-renewable. The resource capacities are, respectively, 4 and 15. The precedence constraints are represented by arrows linking different tasks, as shown in Fig. 1. Table 1 contains the consumptions in resources of the tasks for various modes, as well as their durations.

4.1. Solution representation and initial population

A solution is represented by two vectors: the first one is a position vector which contains the position of each task in the sequence, whereas the second is devoted to the modes assignment. Fig. 2 shows an example of solution where the position vector presents this sequence of tasks {2, 4, 6, 1, 3, 5}.

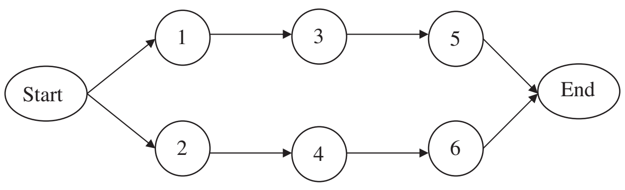

Fig. 1. Example of project.

Table 1
Resource requirements and corresponding durations for two modes

Task	Mode 1		Mode 2	
	Consumption NR/NN	Duration	Consumption NR/NN	Duration
1	2/4	3	1/2	4
2	3/4	4	2/3	6
3	4/2	2	2/2	3
4	4/6	2	3/3	3
5	3/1	1	1/5	3
6	2/1	4	1/1	6

Tasks	1	2	3	4	5	6
Sequences of tasks/C_1^1	4	1	5	2	6	3
Modes of tasks /C_1^1	2	1	2	2	1	1

Fig. 2. Solution representation.

Tasks	1	2	3	4	5	6
Sequences of tasks/ C_0^1	1	4	2	5	3	6
Mode of tasks/ C_0^1	2	1	2	2	1	1
Sequences of tasks/ C_1^1	1	2	3	5	4	6
Mode of tasks/ C_1^1	2	2	1	2	1	1
Sequences of tasks/ C_2^1	2	1	4	3	6	5
Mode of tasks / C_2^1	1	1	1	2	2	2
$rand_1^1$ (sequences)	0.30	0.20	1.00	0.30	0.21	0.10
$rand_1^1$ (modes)	0.57	0.14	0.58	0.60	1.00	0.01
Sequences of tasks/ M_1^1	0.55	4.3	0.50	5.9	2.37	6.15
Mode of tasks / M_1^1	2.85	1.21	2.00	2.00	-0.50	0.98

Fig. 3. Creation of mutant vectors.

The initial population is randomly generated with respect to the precedence constraints.

4.2. Mutant vector

Once the first N_p generation is constructed, we consider three feasible solutions randomly selected C_0^1, C_1^1 and C_2^1. Each solution is represented by the positions of the tasks in the sequence and an assignment of modes for each of them, as shown in Fig. 3. Two vectors are provided for each solution in order to generate the mutant vector by applying (Eq. (2)). In this illustrative example, we just focus on the sequences manipulation; the modes assignments are likely treated with the DE algorithm.

4.3. Trial vector and new solution generation

The trial vectors T_1^1 (positions and modes) are built by copying some elements of the mutant vectors, M_1^1 (positions and modes) into the target vectors C_1^1 (positions and modes), with probabilities equal to Cr_p and Cr_m. As illustrated in Fig. 4, a random number $r_{j,1}^1 \in [0, 1]$ is generated for each element j of each target vector. If $r_{j,1}^1$ (position) $\leqslant Cr_p$, the element of mutant vector (position) is copied, else the element of the target vector T_1^1 (position) is copied, as for the modes vector (Eq. (3)).

For our example, set $Cr_p = 0.2$ and $Cr_m = 0.1$, Fig. 4 shows the production of the trial vector.

With the aim of scheduling the tasks of the new solution, we apply the priority rules to the trial vector. The selection task for each position must satisfy the precedence constraints. Indeed, in our illustrative example, the first task to be scheduled is selected from tasks 1 and 2. Given that the value of task 1 is lower than the value of task 2 (2 < 4.3), then task 1 is certainly scheduled at the first position. The tasks having priority to be scheduled now are tasks 2 and 3. Task 3 gets the second position because 3 < 4.3. Then task 2 competes

Mutant vectors

Sequences of tasks/ M_1^1	0.55	4.3	0.50	5.9	2.37	6.15
Mode of tasks / M_1^1	2.85	1.21	2.00	2.00	-0.50	0.98

Target vectors

Sequences of tasks/ C_1^1	2	1	3	5	4	6
Mode of tasks / C_1^1	2	1	2	2	1	1

r_1^1 (sequences)	0.40	0.14	0.90	0.85	1.00	0.02
r_1^1 (modes)	0.92	0.02	0.80	0.20	0.14	1.00

Trial vectors

Sequences of tasks/ T_1^1	2	4.3	3	5	4	6.15
Mode of tasks / T_1^1	2	1.21	2	2	1	1

Fig. 4. Creation of trial vectors.

Sequences of tasks	1	4	2	5	3	6
Mode of tasks	2	1	2	2	1	1

Fig. 5. Generation of a new solution.

with task 5 to have the third position. Task 5 wins the place since 4 < 4.3 and so on, until all the tasks are scheduled (Fig. 5).

Concerning the modes assignment of the new solution, we determine the round down of the trial vector values between 1 and *m*. Exceptionally, the values which are less than 1 are adjusted to 1 and the values which exceed the maximal number of modes (*m*) are converted to *m*.

4.4. Selection and fitness evaluation

After mutation and crossover operations, the new solution corresponding to the trial vector competes with the current solution corresponding to the target vector for selection into the next generation. The objective function value is used to screen the new solution. If the latter has a better value compared to the current solution, it replaces it in the population, thus allowing the best solution into further generations. This objective value represents the makespan $C_{\max}$ of the sequence. When infeasible solutions can be generated, the fitness function is typically augmented with a penalty function.

Many penalty functions have been implemented in evolutionary computation with several major approaches emerging. The most fundamental is eliminating any infeasible solution from consideration immediately; it is the so called "death penalty". Another approach is based on the number of constraints violated, so that the penalty for a specific number of constraint violations is constant, regardless of the magnitude of the violations. A third, more effective approach uses a distance metric of the infeasible solution from the feasible region which we will use. In other words, the degree of infeasibility of solutions is measured and transformed into a penalty which grows in proportion to the infeasibility level, thus guiding the search toward the feasible space. Consequently, we devise a penalty function to estimate the infeasibility level of a solution, as follows:

$$Penalty = \sum_{l}^{N} \partial.\max\left(0, \frac{\sum_{j=1}^{J} n_{jm_{(j)}l} - N_l}{N_l}\right) \tag{5}$$

Note that $m_{(j)}$ is the mode selected for the task j in the current solution.

The precise definition of the fitness function for infeasible solutions is given below:

$$Fitness = C_{\max} + Penalty \tag{6}$$

5. Implementation and experimental results

The adaptation of the DE algorithm to the MRCPSP is implemented in C++ programming language on Dell desktop PC with Intel Pentium 4 and 3.2 GHz processor.

Recently, in 2003, Bouleimen and Lecocq [6] have solved the MRCPSP using the simulated annealing algorithm. For problems with $n \leqslant 20$, they limited their CPU time per task to 5 s. In order to compare their results to ours, we should fix our time to 150 ms per task. More recently, in 2008, Jarboui et al. [8] have investigated the particle swarm optimization algorithm to solve the MRCPSP and used the same processor; therefore we fixed the same time limit, which is 150 ms per task.

We used a set of standard test problems systematically constructed by the project generator ProGen, which was developed by Kolisch et al. [31]. They are available in the Project Scheduling Problem Library PSPLIB from the University of Kiel.

5.1. Parameter setting

After some experiments, we have fixed the parameters values as follows: the scale factor *A* takes '1.5' in Eq. (2). It is a scaling factor of the difference vector $C_{1,g}$–$C_{2,g}$. *A* has a considerable influence on the exploration, that is, small values of *A* lead to premature convergence, and high values slow down the search. The crossover constants Cr_p and Cr_m are fixed to 0.2 and 0.1, respectively, in Eq. (3), which reflects the probability with which the trial individuals inherit the actual individuals' genes. Although using crossover makes the algorithm rotationally dependent [29,30], crossover becomes desired when we know the properties of an objective function. In Eq. (5), ∂ should be chosen so that the penalty value is greater than the maximal $C_{\max}$ of all solutions in the case of resource constraint violation.

The maximal time per task in ms is fixed to 150.

5.2. Implementation and results

We used benchmark sets for problems with 10, 12, 14, 16, 18 and 20 non-dummy activities, for which only the set of the best known heuristic solutions is available. Each of the non-dummy activities may be performed in one out of three modes. The duration of a mode varies between 1 and 10 periods. We have two renewable and two non-renewable resources. For each problem size, a set of instances was generated by systematically varying four parameters, that is, the resource factor and the resource strength of each resource category. The resource factor is a measure of the average portion of resources requested per task. The resource strength reflects the scarceness of the resources.

For each problem size, a set of 640 instances are generated but, for some instances, no feasible solution exists, therefore these instances are excluded from consideration. Hence, we have 536 instances with $J = 10$, 547 instances with $J = 12$, 551 instances with $J = 14$, 550 instances with $J = 16$, 552 instances with $J = 18$, and 554 instances

Table 2
Solutions for MRCPSP benchmark instances with 150 ms/activity

Tasks number	Feasible solutions	Simulated annealing		Particle swarm optimization		Differential evolution		
		Average deviation	Percentage of optima found	Average deviation	Percentage of optima found	Maximal deviation	Average deviation	Percentage of optima found
10	536	0.21	96.30	0.06	99.25	0.15	0.09	99.30
12	547	0.19	91.20	0.14	98.47	0.29	0.11	99.30
14	551	0.92	82.60	0.44	91.11	0.63	0.34	97.60
16	550	1.43	72.80	0.59	85.91	0.83	0.42	96.38
18	552	1.85	69.40	0.99	79.89	1.58	0.59	94.43
20	554	2.10	66.90	1.21	74.19	1.91	0.70	91.75

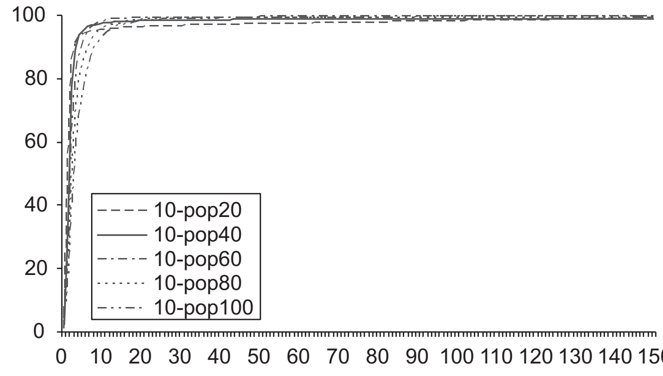

Fig. 6. Effect of the population size on percentage of problems solved to optimality (instances with 10 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

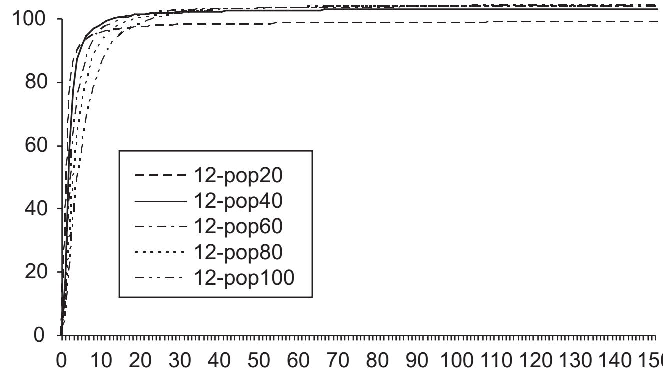

Fig. 7. Effect of the population size on percentage of problems solved to optimality (instances with 12 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

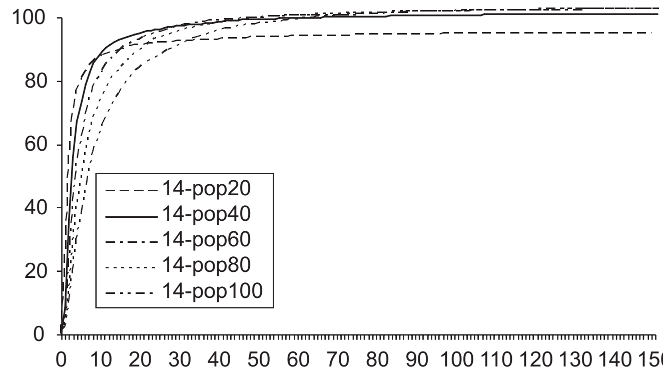

Fig. 8. Effect of the population size on percentage of problems solved to optimality (instances with 14 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

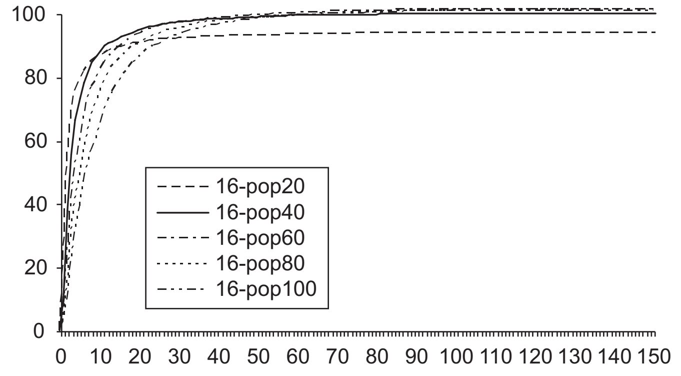

Fig. 9. Effect of the population size on percentage of problems solved to optimality (instances with 16 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

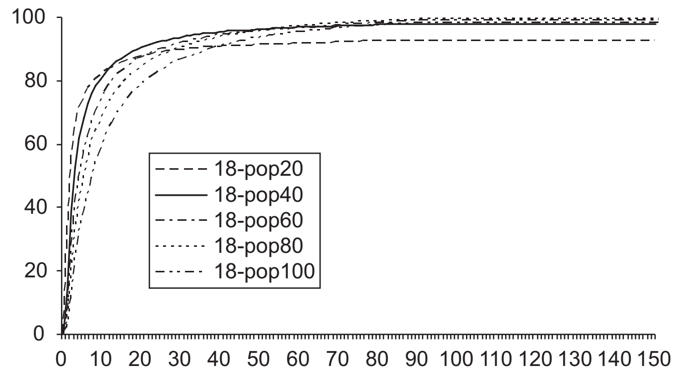

Fig. 10. Effect of the population size on percentage of problems solved to optimality (instances with 18 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

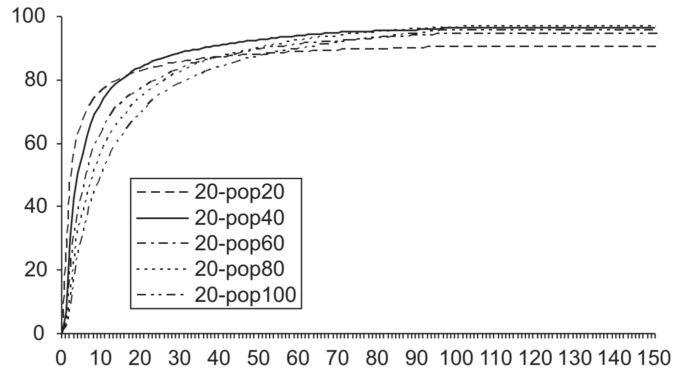

Fig. 11. Effect of the population size on percentage of problems solved to optimality (instances with 20 tasks). The vertical axis represents the percentage of problems solved to optimality and the horizontal axis represents the time in ms.

Table 3
Solutions for MRCPSP benchmark instances with five different population sizes and 15 different times/task in ms

Instances	Population size	Time in ms														
		10	20	30	40	50	60	70	80	90	100	110	120	130	140	150
10	20	95.3	96.3	96.7	96.9	97.2	97.4	97.6	97.8	98.0	98.2	98.3	98.5	98.7	98.9	99.1
	40	97.6	98.3	98.5	98.6	98.7	98.7	98.7	98.7	98.8	98.8	98.8	98.8	98.8	98.8	98.8
	60	97.6	99.1	99.3	99.3	99.4	99.4	99.4	99.4	98.4	99.4	99.4	99.8	99.4	99.4	99.4
	80	95.9	98.6	99.0	99.2	99.3	99.3	99.3	99.3	99.3	99.3	99.3	99.3	99.3	99.3	99.3
	100	93.3	98.3	98.8	99.0	99.1	99.1	99.1	99.3	99.4	99.4	99.4	99.4	99.4	99.4	99.4
12	20	91.5	93.1	93.7	93.8	93.9	94.1	94.1	94.2	94.3	94.3	94.4	94.4	93.4	94	94.4
	40	94.2	96.9	97.6	97.9	98.1	98.1	98.3	98.3	98.4	98.4	98.5	98.5	98.5	98.5	98.5
	60	92.4	96.9	98.1	98.4	98.7	98.8	98.9	99.0	99.0	99.1	99.1	99.1	99.1	99.1	99.1
	80	89.2	95.9	97.5	98.3	98.7	99.0	99.1	99.1	99.2	99.2	99.3	99.3	99.3	99.3	99.3
	100	83.1	94.2	97.1	98.2	98.6	98.9	99.2	99.3	99.4	99.4	99.5	99.5	99.5	99.5	99.5
14	20	83.6	87.0	88.1	88.7	89.1	89.5	89.7	89.7	90.0	90.2	90.3	90.4	90.4	90.4	90.4
	40	84.9	90.7	92.7	94.1	94.6	95.0	95.4	95.4	95.7	95.9	96.0	96.1	96.2	96.3	96.3
	60	78.9	89.1	92.9	94.5	95.2	95.6	95.9	95.9	96.7	97.1	97.2	97.3	97.5	97.6	97.6
	80	71.4	86.2	91.4	93.7	95.2	95.8	96.4	96.4	96.9	97.1	97.2	97.4	97.6	97.6	97.6
	100	62.9	80.8	87.8	91.7	93.5	94.9	95.7	95.7	96.8	97.1	97.3	97.5	97.7	97.8	97.8
16	20	82.9	86.8	88.0	88.6	88.8	89.1	89.3	89.5	89.5	89.5	89.5	89.5	89.5	89.5	89.5
	40	84.2	90.8	93.0	93.8	94.4	94.8	95.0	95.2	95.3	95.4	95.4	95.4	95.4	95.4	95.4
	60	78.7	89.7	92.8	94.1	95.0	95.6	96.2	96.3	96.4	96.5	96.5	96.5	96.5	96.5	96.5
	80	72.6	86.9	91.2	93.5	94.2	94.9	95.6	96.0	96.1	96.3	96.3	96.3	96.3	96.3	96.3
	100	62.6	82.9	89.6	92.5	94.0	94.9	95.3	95.8	96.1	96.1	96.1	96.1	96.1	96.1	96.1
18	20	78.7	83.6	85.4	86.3	86.8	87.2	87.7	88.0	88.1	88.1	88.1	88.1	88.1	88.1	88.1
	40	78.0	86.3	89.4	90.9	91.6	92.1	92.6	93.0	93.1	93.1	93.1	93.1	93.1	93.1	93.1
	60	72.5	83.6	87.7	90.1	91.1	92.0	92.9	93.6	94.0	94.0	94.0	94.0	94.0	94.0	94.0
	80	67.0	81.1	86.5	89.6	91.4	92.7	93.5	94.0	94.4	94.4	94.4	94.4	94.4	94.4	94.4
	100	58.6	75.9	82.9	87.0	89.4	91.0	92.1	92.9	93.3	93.3	93.3	93.3	93.3	93.3	93.3
20	20	80.2	86.4	89.2	91.0	91.8	92.5	93.0	93.4	93.8	94.1	94.1	94.1	94.1	94.1	94.1
	40	70.2	80.2	84.4	86.4	88.1	89.4	90.3	90.6	91.3	91.4	91.4	91.4	91.4	91.4	91.4
	60	61.1	73.6	79.9	83.0	85.3	86.8	87.7	88.6	89.3	89.6	89.6	89.6	89.6	89.6	89.6
	80	55.5	71.2	79.0	83.3	85.9	88.1	89.4	90.4	91.3	91.7	91.7	91.7	91.7	91.7	91.7
	100	48.6	66.9	75.4	80.3	83.4	85.6	87.5	89.0	90.2	90.9	90.9	90.9	90.9	90.9	90.9

with $J = 20$. The set with 20 non-dummy activities currently is the hardest standard set of multi-mode instances for which all optimal solutions are known, cf. Sprecher and Drexl [1]. For the set with 30 activities, we have only 540 feasible instances and not all optimal solutions are known, so we did not consider it.

5.2.1. *Competitiveness of the method*

The results of the experiments are presented in Table 2. For each problem size we indicate the percentage of problems solved to optimality, the average and the maximal deviations from the optimality.

Comparison of our results with those provided by simulated annealing and particle swarm optimization algorithms confirms that DE is widely more competitive, for all problems sizes.

In fact, our percentages of problems solved to optimality and average deviations are better than those of the two other algorithms. Concerning the maximal deviations of our algorithm, we can observe that they are better than the average deviations of the simulated annealing algorithm.

5.2.2. *Effect of population size*

In this sub-section, the effect of the population size and the time per task on the performance of DE is investigated. Figs. 6–11 summarize the results obtained by using five different population sizes in fifteen different times per activity to calculate the percentage of problems solved to optimality. The details are given in Table 3. It is worth noting that the population size is a significant parameter in the DE algorithm which can affect the solution quality. The stochastic feature of the method obliged us to run the algorithm 10 times in order to have the mean percentage of problems solved to optimality. We have saved the results obtained at each ms, which explains the form of the curves illustrated in Figs. 6–11. We observe that the results of 20-sized population are usually the best only with small time (5 ms). From about 10–40 ms, we note that the curve corresponding to population 40 is above the four remaining curves. In average, the percentage of problems solved to optimality for all instances improves from 40 ms with population sizes 60, 80 and 100 but they are not far from 40-sized population, which presents both features of exploration and exploitation of the method. This can be explained by the fact that high sized-population needs more time to perform.

6. Conclusions

The resolution of the MRCPSP by DE algorithm is proposed in this paper. We compared our results with those obtained through the simulated annealing algorithm of Bouleimen and Lecocq [6] and the particle swarm optimization of Jarboui et al. [8]. Table 2 shows that DE strictly dominates all other alternatives. In addition, we analysed the effect of the population size on the solution quality. As a conclusion, we noted that the population size is an important parameter which has a high influence on the solution quality.

In addition to the population size, DE algorithm presents more parameters, which can be analysed in order to deduce the best alternative for our problem. In other words, we could search for another mutation strategy, more efficient for the MRCPSP.

References

[1] Sprecher A, Drexl A. Solving multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. European Journal of Operational Research 1998;107:431–50.

[2] Drexl A, Grünewald J. Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 1993;25:74–81.

[3] Slowinski R, Soniewicki B, Weglarz J. DSS for multiobjective project scheduling subject to multiple-category resource constraints. European Journal of Operational Research 1994;79:220–9.

[4] Kolisch R, Drexl A. Local search for non-preemptive multi-mode resource-constrained project scheduling. IIE Transactions 1997;29:987–99.

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

[6] Bouleimen K, Lecocq H. A new efficient simulated annealing algorithm for the resource constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 2003;149:268–81.

[7] Hartmann S, Drexl A. Project scheduling with multiple modes: a comparison of exact algorithms. Networks 1998;32:283–97.

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

[9] Boctor FF. Heuristics for scheduling projects with resource restrictions and several resource duration modes. International Journal of Production Research 1993;31:2547–58.

[10] Boctor FF. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research 1996;90:349–61.

[11] Storn R, Price K. System design by constraint adaptation and differential evolution. IEEE Transactions on Evolutionary Computation 1999;3(1): 22–34.

[12] Bergey PK. An agent enhanced intelligent spreadsheet solver for multi-criteria decision making. In: Proceedings of AIS AMCIS 1999: American conference on information systems. Milwaukee, USA; 1999. p. 966–8.

[13] Babu BV, Sastry KKN. Estimation of heat transfer parameters in a trickle-bed reactor using differential evolution and orthogonal collocation. Computers and Chemical Engineering 1999;23(3):327–39.

[14] Rajive J, Arthur C. Minimal representation multisensor fusion using differential evolution. IEEE Transactions on Systems, Man and Cybernetics, Part A 1999;29(1):63–76 ISSN: 1083-4427.

[15] Jouni L, Ivan Z. Mechanical engineering design optimization by differential evolution. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London, UK: McGraw-Hill; 1999. p. 127–46 ISBN 007-709506-5.

[16] Storn R. System design by constraint adaptation and differential evolution. IEEE Transactions on Evolutionary Computation 1999;3(1):22–34.

[17] Chang T, Chang H. An efficient approach for reducing harmonic voltage distortion in distribution systems with active power line conditioners. IEEE Transactions on Power Delivery 2000;15(3):990–5.

[18] Lameh J, Wang P, Elgart D, Meredith D, Shafer SL, Loew GH. Unraveling the identity of benzodiazepine binding sites in rat hippocampus and olfactorybulb. European Journal of Pharmacology 2000;400(2–3):167–76.

[19] Rogalsky T, Kocabiyik S, Derksen RW. Differential evolution in aerodynamic optimization. Canadian Aeronautics and Space Journal 2000;46(4):183–90.

[20] Stumberger G, Pahner U, Hameyer K. Optimization of radial active magnetic bearings using the finite element technique and the differential evolution algorithm. IEEE Transactions on Magnetics 2000;36(4):1009–13.

[21] Wang F, Sheu J. Multiobjective parameter estimation problems of fermentation processes using a high ethanol tolerance yeast. Chemical Engineering Science 2000;55(18):3685–95.

[22] Booty WG, Lam CLD, Wong IWS, Siconolfi P. Design and implementation of an environmental decision support system. Environmental Modelling and Software 2000;16(5):453–8.

[23] Cheng S, Hwang C. Optimal approximation of linear systems by a differential evolution algorithm. IEEE Transactions on Systems, Man and Cybernetics, Part A 2001;31(6):698–707.

[24] Hendtlass T. A combined swarm differential evolution algorithm for optimization problems. In: Lecture notes in computer science, vol. 2070. Berlin: Springer; 2001.

[25] Ruzek B, Kvasnicka M. Differential evolution algorithm in the earthquake hypocenter location. Pure and Applied Geophysics 2001;158(4):667–93.

[26] Abbass HA. An evolutionary artificial neural networks approach for breast cancer diagnosis. Artificial Intelligence in Medicine 2002;25:265–81.

[27] Abbass HA. A memetic Pareto evolutionary approach to artificial neural networks. In: Lecture notes in artificial intelligence, vol. 2256. Berlin: Springer; 2002.

[28] Babu BV, Jehan MML. Differential evolution for multi-objective optimization. In: Proceedings of 2003 conference on evolutionary computation (CEC-2003). Canberra, Australia; 2003. p. 2696–703.

[29] Lorenzoni LL, Ahonen H, Alvarenga AG. A multi-mode resource-constrained scheduling problem in the context of port operations. Computers and Industrial Engineering 2006;50:55–6.

[30] Storn R, Price K. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI; March 1995.

[31] Kolisch R, Sprecher A, Drexl A. Characterization and generation of a general class of resource constrained project scheduling problems. Management Science 1995;41:1693–703.