J Heuristics (2006) 12:395–411
DOI 10.1007/10732-006-3750-x

Differential evolution for sequencing and scheduling optimization

Andreas C. Nearchou · Sotiris L. Omirou

Submitted in April 2005 and accepted by Zbigniew Michalewicz in November 2005 after 1 revision

Abstract This paper presents a stochastic method based on the differential evolution (DE) algorithm to address a wide range of sequencing and scheduling optimization problems. DE is a simple yet effective adaptive scheme developed for global optimization over continuous spaces. In spite of its simplicity and effectiveness the application of DE on combinatorial optimization problems with discrete decision variables is still unusual. A novel solution encoding mechanism is introduced for handling discrete variables in the context of DE and its performance is evaluated over a plethora of public benchmarks problems for three well-known NP-hard scheduling problems. Extended comparisons with the well-known random-keys encoding scheme showed a substantially higher performance for the proposed. Furthermore, a simple slight modification in the acceptance rule of the original DE algorithm is introduced resulting to a more robust optimizer over discrete spaces than the original DE.

Keywords Differential evolution · Encoding solutions · Representation mechanism · Discrete optimization · Scheduling · Meta-heuristics

1 Introduction

Differential evolution (DE) is a population-based heuristic recently proposed by Storn and Price (1997) for global optimization over continuous spaces. DE codes the space to be searched as a set of D-dimensional floating-point vectors. Initially, a population of N randomly chosen vectors is created, and DE's attempt is to replace all the vectors of the population, with better ones (generated by specific variation operators) per iteration. The overall iterative process cycles until a suitable, user-defined stopping criterion is satisfied.

A. C. Nearchou (✉)
Department of Business Administration, University of Patras, 26 500 Patras, Greece
e-mail: nearchou@upatras.gr

S. L. Omirou
Department of Mechanical Engineering, Frederick Institute of Technology, Nicosia, Cyprus
e-mail: eng.os@fit.ac.cy

Since its invention, DE has been applied with high success on many numerical optimization problems (Cheng and Hwang, 2001; Lin et al., 2004; Sun et al., 2005). Moreover, different variants of the basic DE model have been proposed to improve its efficiency over continues problems. Ali and Törn (2004) developed several variants of the basic DE. Comparisons showed that these variants are superior to the original DE, as well as, to many other direct search global optimization algorithms including genetic algorithms (GAs), and the controlled random search method. Kaelo and Ali (2006) proposed modifications in mutation and localization in the acceptance rule of DE resulting in two new versions of the original algorithm. Experiments over multiple benchmark test problems showed that these new DE algorithms are superior to the original DE. A thorough comparative study of the performance of DE and four other global optimization algorithms is presented in Ali et al. (2005). The methodology used by the authors based on a modified performance profile plot together with quartile sequential plots is general and applicable to stochastic algorithms.

Despite the simplicity and the high efficiency of DE, its application on the solution of combinatorial optimization problems (COPs) with discrete decision variables is still unusual. One of the possible reasons for this lack is its difficulty to evolve permutation strings so that to directly present feasible solutions to these problems. Note that, most of these problems have solutions presented through permutation vectors, while DE maintains and evolves floating-point vectors. In the framework of population-based heuristics, a well-known representation technique that deals with floating-point vectors for discrete optimization of COPs is the random-keys encoding (Bean, 1994); proposed to work with real-coded GAs. After experimentation, we found random-keys encoding not well suited within DE for discrete optimization, and thus it was decided to implement a new robust encoding scheme.

The purpose of this paper is twofold: first, to investigate the application of the DE on sequencing and scheduling COPs. To the best of our knowledge, very few works such as (Onwubolu, 2004; Nearchou, 2006), have applied DE on COPs, such as the traveling salesman problem (Onwubolu, 2004), and the machine layout design problem (Nearchou, 2006). The majority of previous works studied the application of DE on global optimization over continuous spaces only. To that purpose, a novel representation scheme is introduced that can be used by DE for vector's encoding. This scheme maps real-coded vectors to feasible permutation strings thus giving the ability to DE to address any COP that follows the permutation property.[1] As a second contribution of this paper, we suggest a very simple (yet effective) modification in the acceptance rule of the original DE algorithm. Extended numerical studies showed that this modification results to a much more robust optimizer over discrete spaces than the original DE algorithm.

The performance of the proposed DE is evaluated through experimental work over public benchmark test problems with known lower (or upper) bounds for three classic scheduling problems: the flow-shop scheduling problem (FSSP), the single-machine total weighted tardiness problem (TWTP), and the single machine common due date scheduling problem (CDDSP). Three different DE strategies are examined each one based on a different exploration/exploitation mechanism.

The rest of this paper is organized as follows: Section 2 gives a brief description of the basic DE model for optimization over continuous spaces. Section 3 formulates the three COPs (FSSP, TWTP, CDDSP) under consideration. Section 4 introduces a new solutions' encoding scheme that can effectively address a wide range of COPs with discrete decision variables. A

[1] The solutions to these problems are represented by all the possible permutations $\{\pi_1, \pi_2, \ldots, \pi_n\}$ of a sequence of integers $\{1, 2, \ldots, n\}$ who correspond to the decision variables of the problem.

step-by-step description of the proposed DE algorithm for discrete optimization is presented in Section 5. Section 6, presents and discusses in detail comparative experimental results of the algorithms over public benchmarks concerning the three scheduling problems. Moreover, in the same section a very simple yet effective DE variant is introduced and incorporated in the overall experimental process. Finally, Section 7 summarizes the contribution of the paper and states some directions for future work.

2 DE: The basic model

The DE algorithm utilizes *N*, *D*-dimensional parameter vectors $x_{i,k}$, $i = 1, 2, \ldots, N$, as a population to search the feasible region Ω. The index *k* denotes the iteration (or generation) number of the algorithm. The initial population,

$$S = \{x_{1,0}, x_{2,0}, \ldots, x_{N,0}\}, \tag{1}$$

where $k = 0$, is taken to be uniformly distributed in the search region. At each iteration, all vectors in *S* are targeted for replacement. Therefore, *N* competitions are held to determine the numbers of *S* for the next iterations. This is achieved by using mutation, crossover and acceptance operators. In the mutation phase, for each target vector $x_{i,k}$, $i = 1, 2, \ldots N$, a mutant vector $\widehat{x}_{i,k}$ is obtained by

$$\widehat{x}_{i,k} = x_{\alpha,k} + F(x_{\beta,k} - x_{\gamma,k}), \tag{2}$$

where $\alpha, \beta, \gamma \in \{1, 2, \ldots, N\}$ are mutually distinct random indices and are also different from the current target index *i*. The vector $x_{\alpha,k}$ is known as the base vector and $F > 0$ is a scaling parameter. The crossover operator is then applied to obtain the trial vector $y_{i,k}$ from $\widehat{x}_{i,k}$ and $x_{i,k}$. The crossover is defined by

$$y_{i,k}^{j} = \begin{cases} \widehat{x}_{i,k}^{j} & \text{if } R^j \leq C_R \quad \text{or } j = I_i \\ x_{i,k}^{j} & \text{if } R^j > C_R \quad \text{and } j \neq I_i \end{cases}, \tag{3}$$

where I_i is a randomly chosen integer in the set *I*, i.e., $I_i \in I = \{1, 2, \ldots, D\}$; the superscript *j* represents the *j*-th component of respective vectors; $R^j \in (0, 1)$, drawn randomly for each *j*. The ultimate aim of the crossover rule is to obtain the trial vector $y_{i,k}$ with components coming from the components of the target vector $x_{i,k}$ and the mutated vector $\widehat{x}_{i,k}$. This is ensured by introducing C_R and the set *I*. Notice that for $C_R = 1$ the trial vector $y_{i,k}$ is the replica of the mutated vector $\widehat{x}_{i,k}$. The targeting process (mutation and crossover) continues until all members of *S* are considered. After all *N* trial vectors $y_{i,k}$ have been generated, acceptance is applied. In the acceptance phase, the function value at the trial vector, $f(y_{i,k})$, is compared to $f(x_{i,k})$, the value at the target vector and the target vector is updated using

$$x_{i,k+1} = \begin{cases} y_{i,k} & \text{if } f(y_{i,k}) < f(x_{i,k}) \\ x_{i,k} & \text{otherwise} \end{cases} \tag{4}$$

Reproduction (mutation and crossover) and acceptance continues until some stopping conditions are met.

The mechanism described above is only one variant of the basic DE algorithm known as scheme DE1 (Storn and Price, 1997). There are also some other DE variants, only differ in the way they create the mutant vector (Equation (2)). Two such well-known variants called DE2 and DE3, respectively are given by the following two relations:

$$\widehat{x}_{i,k} = x_{\alpha,k} + \lambda(x_{\text{best},k} - x_{\alpha,k}) + F(x_{\beta,k} - x_{\gamma,k}) \tag{5}$$

Where, λ is a control variable and $x_{\text{best},k}$ is the best vector of the current population.

$$\widehat{x}_{i,k} = x_{\text{best},k} + F(x_{\alpha,k} + x_{\beta,k} - x_{\gamma,k} - x_{\delta,k}) \tag{6}$$

In Eq. (6) the mutant vector is generated using four vectors randomly selected from the population (with $\alpha \neq \beta \neq \gamma \neq \delta \neq i \in \{1, 2, \ldots, N\}$ together with the population best vector. This scheme is known as DE3. The usage of two difference vectors seems to improve the diversity of the population.

3 Sequencing and scheduling problems

3.1 The flow-shop scheduling problem (FSSP)

FSSP is a strongly NP-hard (Rinnooy, 1976) COP that has captured the interest of a significant number of researchers. The problem can be described as follows: let n jobs from a job set $\{1, 2, \ldots, n\}$, with $n > 1$, have to be processed on m machines $\{1, 2, \ldots, m\}$ in the order given by the indexing of the machines. Each job consists of m operations and each operation requires a different machine. The processing time of each job i on machine j is fixed and denoted by t_{ij} $(i = 1, \ldots, n; j = 1, \ldots, m)$. Preemption is not allowed, thus, the operation of each job on a machine requires an uninterrupted period of time. Each job can be processed on one machine at a time and each machine can process only one job at a time. The objective is to find the optimal schedule (permutation of all jobs), which has the minimum sum of job completion times.

Let $C(j_i, k)$ denotes the completion time of job j_i on machine k, and let $\{j_1, j_2, \ldots, j_n\}$ denotes a permutation of jobs, then the completion time for an n-job m-machine FSSP is calculated as follows:

$$\begin{aligned}
&C(j_1, 1) = t_{j_1 1}\\
&C(j_1, k) = C(j_1, k-1) + t_{j_1 k}, \quad k = 2, \ldots, m\\
&C(j_i, 1) = C(j_{i-1}, 1) + t_{j_i 1}, \quad i = 2, \ldots, n\\
&C(j_i, k) = \max\{C(j_{i-1}, k), C(j_i, k-1)\} + t_{j_i k}, \quad i = 2, \ldots, n, k = 2, \ldots, m
\end{aligned} \tag{7}$$

Then, the FSSP can be formulated by

$$\min C_{\max} = C(j_n, m). \tag{8}$$

3.2 The total weighted tardiness problem (TWTP)

TWTP is an NP-hard problem for which instances with more than 50 jobs cannot be solved to optimality with state-of-the-art enumeration techniques such as branch and bound algorithms

(Abdul-Razaq et al., 1990). In TWTP, n jobs have to be processed on a single machine. Associated with each job j there are three quantities: a processing time p_j, a weight w_j (denoting the relative priority of job j), and a due date d_j $(j = 1, 2, \ldots, n)$. The jobs are available for processing at time zero, while the machine is available for continuous processing of the jobs. Once a job begins processing it is completed without interruption (i.e. preemption is not allowed). The tardiness of a job j is defined as $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of job j in the current job sequence. The goal is to find a sequence of jobs which minimizes the sum of the weighted tardiness, i.e., TWTP can be formally written as

$$\min \sum_{j=1}^{n} w_j \cdot T_j \tag{9}$$

3.3 The common due date scheduling problem (CDDSP)

CDDSP can be formally defined as follows: consider n jobs (numbered $1, 2, \ldots, n$) to be processed without interruption on a single machine which can handle only one job at a time. Each job j $(j = 1, \ldots, n)$ is available at time zero, requires a positive processing time p_j and ideally must be completed exactly on a specific (common for all jobs) due date d. Penalties are incurred whenever a job is completed before or after this due date. Therefore, an ideal schedule is one in which all jobs finish on the specific due date. Assuming that C_j is the completion time of job j, then the earliness and tardiness of job j are given by the relations, $E_j = \max(0, d - C_j)$ and $T_j = \max(0, C_j - d)$, respectively, for all $j = 1, \ldots, n$. The objective is therefore to find a processing order of the n jobs that minimizes

$$\sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j) \tag{10}$$

where, α_j, β_j $(j = 1, \ldots, n)$ are the earliness and tardiness (nonnegative) penalties, respectively, for job j and constitute data input to the scheduling problem. A common due date d is called unrestrictive when $d \geq \sum_1^n p_j$ holds, otherwise is called restrictive. Even the simplest formulation CDDSP leads to an NP-hard COP (Baker and Scudder 1990). In this work we consider one of the hardest versions of the basic CDDSP, that with a restrictive common due date and different earliness/tardiness penalties.

4 Representation schemes for sequencing and scheduling optimization problems

4.1 The random-keys encoding

Representation (or coding) is the mapping from a state space of possible solutions to a state space of encoded solutions within a particular data structure. The natural coding for sequencing and scheduling problems is the permutation vectors (or integer vectors). So the solution (1 6 4 3 7 2 5) for a 7-job FSSP represents the order in which the jobs are executed on a number of m machines, i.e., job 1 followed by job 6, followed by job 4, etc. Similarly, the same solution for a 7-job TWTP, or a 7-job CDDSP represents the order in which the 7 jobs must proceed on a specific single machine with objective the minimization of the functions given in Eqs. (9), or (10), respectively.

In order to apply the DE algorithm on such COPs, it is crucial to design a suitable encoding scheme that maps the floating-point vectors to good permutation vector solutions. Random-keys representation (Bean, 1994) is the most famous solutions' encoding scheme in the field of population heuristics dealing with sequencing optimization problems. According to this technique, each solution is encoded as a vector of D floating-point numbers, where D corresponds to the number of problem's parameters. The components of the vector are sorted and their order in the vector determines the final solution to the problem. For example, for a 7-job FSSP (alternatively TWTP, or CDDSP), the vector (0.23, 0.82, 0.03, 0.47, 0.21, 0.15, 0.68) corresponds to the job sequence 3 – 6 – 5 – 1 – 4 – 7 – 2, since the 3rd component in the vector has the lowest value (=0.03), followed by the 6th component in the vector, which is the second smallest number (=0.15), and so on. Note that the application of any traditional crossover on the floating-point vectors always results to legal solutions to the physical problem and thus, this method overcomes the difficulty of maintaining feasibility from parent to offspring appeared with permutation encoding. Although random-keys work well in the context of a real-coded GA (Bean, 1994), after much experimentation we found this encoding scheme to be poor suited within the DE algorithm to address COPs.

4.2 The sub-range encoding: A new solution representation scheme for COPs

Due to the low performance of DE with random-keys, a new solutions encoding mechanism was designed and used. The main features of the proposed representation technique are described below. Without lose the generality, in the description we will use terms borrowed from the field of Evolutionary Computation such as *the genotype* (i.e., the vector's structure evolved by the DE algorithm), *the phenotype* (i.e., the actual solution to the physical problem corresponding to a specific genotype). Accordingly, every component of a vector is called *gene*.

1. In a pre-processing phase, the range $[1, D]$ (where D is the number of problem's parameters) is divided into D equal sub-ranges and the upper bound of each sub-range is saved in an array of floating-point numbers. Let's call this array *SR* (stands for Sub-Range). Therefore, the content of the array is $SR = [1/D, 2/D, 3/D, \ldots, D/D]^T$.
2. Each floating-point vector in the genotypic level is encoded as a D-dimensional real-valued vector with each gene corresponding to a decision variable of the physical COP.
3. Each genotype is mapped to a corresponding phenotype. The components of a phenotype are integer numbers in [1, D]; sorted according to the sub-range index in which belong the corresponding genes of the genotype.
4. Crossover and mutation operators are performed in the genotypic level, not on the derived solutions (i.e., not on the phenotypes).
5. Each phenotype is then checked and repaired so that to finally present a valid solution to the COP.

The mechanism of building the proto-phenotype of a given genotype g works as follows:

```
Procedure: **Proto-Phenotype** (SR, $g$)
Step 1) Let $j = 1$    // $j$ denotes the position of the gene in the genotype $g$ //
Step 2) Determine the sub-range index corresponding to the $j$-th gene of the
   vector. Let's $q$ ($q \in I = 1, 2, \ldots, D$) the index of this sub-range.
Step 3) Put the integer $q$ in the $j$-th position of the proto-phenotype solution $\mathbf{P}^g$
Step 4) Let $j = j + 1$.
Step 5) Repeat steps (2)–(4) until $j > D$.
Step 6) Return ($\mathbf{P}^g$)
```

Table 1 Building phenotypes from real-coded genotypes via the sub-range encoding scheme

Gene position	Gene value	Sub-range index	Generated proto-phenotype
1	0.23	2	(2 _ _ _ _ _ _)
2	0.82	6	(2 6 _ _ _ _ _)
3	0.03	1	(2 6 1 _ _ _ _)
4	0.47	4	(2 6 1 4 _ _ _)
5	0.21	1	(2 6 1 4 1 _ _)
6	0.15	2	(2 6 1 4 1 2 _)
7	0.68	5	(2 6 1 4 1 2 5)

For example, let again the genotype, $g = (0.23, 0.82, 0.03, 0.47, 0.21, 0.15, 0.68)$ mentioned in Section 4.1. Since the related COP has 7 decision variables ($D = 7$), then the array $SR = [1/7, 2/7, 3/7, 4/7, 5/7, 6/7, 7/7]^T = [0.14, 0.29, 0.43, 0.57, 0.71, 0.86, 1.0]^T$. Table 1 shows analytically how the phenotype corresponding to g is built using the above procedure.

As one can see from Table 1, the first gene (=0.23) lies in the second sub-range ($0.14 < 0.23 \leq 0.29$), the second gene (=0.82) lies in the sixth sub-range ($0.71 < 0.82 \leq 0.86$), etc. As it is clear, the generated final phenotype (2 6 1 4 1 2 5) is illegal since it contains duplicated genes. Hence, to finally produce a valid version of the phenotype vector the following two-steps repairing procedure it is applied on the proto-phenotype:

1. Delete the duplicate genes.

   (2 6 1 4 _ _ 5)

2. Fill the empty locations in the vector by selecting randomly the remaining (unused) genes.

   (2 6 1 4 **7** **3** 5)

Therefore, the genotype $g = (0.23, 0.82, 0.03, 0.47, 0.21, 0.15, 0.68)$ corresponds to the job sequence 2 – 6 – 1 – 4 – 7 – 3 – 5; which is different from the job-sequence generated by the random-keys encoding (see sub-section 4.1).

5 The DE algorithm for discrete COPs

Since we described the proposed representation mechanism for mapping floating-point vectors to actual scheduled solutions we are now ready to present step-by-step the proposed DE implementation for discrete optimization.

```
Algorithm DE for discrete optimization
Pre-processing step:
    Read all the necessary input data related to the specific COP;
    Build the Sub-Range array: $SR = [1/D, 2/D, 3/D, \ldots, 1]^T$;
Initialization step:
    Set values for the DE control parameters ($N$, $F$, $C_R$);
    Initialize generation counter $k = 0$;
    Generate a population $S$ with $N$, $D$-dimensional floating-point vectors
        $S = \{x_{1,k}, x_{2,k}, \ldots, x_{N,k}\}$;
    // where the components of each point $x_{i,k}$, $i = 1, 2, \ldots N$, are floating-point
      numbers randomly chosen within the range [0,1]. //
```

```
Repeat
  for $i = 1$ to $N$ do
    Mutation step:
      Generate a mutant vector $\widehat{x}_{i,k}$ using Equation (2).
    Crossover step:
      Generate a trial vector $y_{i,k}$ by crossing $x_{i,k}$ and $\widehat{x}_{i,k}$using Equation (3).
    Solution Interpretation Step:
      // build the proto-phenotypes corresponding to the genotypes $x_{i,k}$ and $y_{i,k}$ //
      $\mathbf{P}^{x_{i,k}} \leftarrow$ Proto_Phenotype (SR,$x_{i,k}$);
      $\mathbf{P}^{y_{i,k}} \leftarrow$ Proto_Phenotype (SR,$y_{i,k}$);
      // repair (if needed) the phenotypes so that to correspond to feasible
        solutions //
      Repair ($\mathbf{P}^{x_{i,k}}$); Repair ($\mathbf{P}^{y_{i,k}}$);
    Acceptance step:
      if COST($\mathbf{P}^{y_{i,k}}$) $<$ COST($\mathbf{P}^{x_{i,k}}$) then $x_{i,k+1} = y_{i,k}$else $x_{i,k+1} = x_{i,k}$
endfor
  Population Statistics Step:
    Determine the population best phenotype solution $\mathbf{P}^{\text{best}}$;
    if $k = 0$ then
      $\mathbf{P}^* = \mathbf{P}^{\text{best}}$      // keep track for the best-so-far solution //
    else if COST($\mathbf{P}^{\text{best}}$) $<$ COST($\mathbf{P}^*$) then
      $\mathbf{P}^* = \mathbf{P}^{\text{best}}$
endif
    $k = k + 1$      // increment iteration counter //
Until $k >$ MAXI; // MAXI stands for Maximum Iterations //
Return ($\mathbf{P}^*$)
```

In a pre-processing phase, the algorithm takes as input, data concerning the specific COP to be solved. Specifically, the input information include: (a) for FSSP the number of jobs (n) to be scheduled, the number of operations (m) for each job, and the processing time per job. (b) For TWTP, the number of jobs, the processing time and the due date per job, and a weight factor denoting the relative priority of each job. (c) For CDDSP, the number of jobs to be scheduled, a due date common for all jobs, and for each job the processing time, and two penalty coefficients related to the earliness or the tardiness of the job. All the problems are of dimension $D = n$.

The mutation and crossover steps are identical to the original DE algorithm described in Section 2 and responsible to create the mutant and trial vectors, respectively. The solution interpretation step is responsible to generate for these target and trial genotypes the corresponding phenotypes. This is done by calling the procedures *Proto_Phenotype()* and *Repair()* described in Section 4.2. The resulting phenotypes represent actual solutions to the COP under consideration. The quality of these solutions is evaluated by function COST() in the acceptance step of the algorithm. In particular, COST() computes the actual cost of each phenotype solution. For the case of FSSP, this cost corresponds to the makespan ($C_{\max}$) (Equation (8)), while for TWTP and CDDSP, this cost correspond to the summations given by Equations (9) and (10), respectively. The genotype corresponding to the best phenotype (i.e., to the solution with the lowest cost) survives and replaces the current target vector of the population. The four steps (mutation, crossover, solution interpretation, acceptance) are repeated for all the N points of the entire population.

After generating the population of the next iteration, the algorithm determines the best population phenotype. The performance of this solution is then compared to that of the best-so-far solution. The overall process is repeated until a maximum number of iterations (*MAXI*) is surpassed. The returned by the algorithm best-so-far solution constitutes the 'best' solution to the COP.

6 Experimental results and discussion

6.1 Comparative heuristics

The effectiveness of the DE algorithm was examined through simulation experiments over public benchmarks instances for the three COPs presented in Section 3. Three DE variants namely, DE1, DE2 (with $\lambda = 0.5$), and DE3 (all described in Section 2) were implemented and evaluated, first using the proposed sub-range solution encoding scheme and then using the random-keys encoding. In the following analysis we will refer to the former scheme by SR and to the latter by Rkeys. Accordingly, we will refer to the various versions of DE with notations such as DE3_SR (stands for DE3 with SR), DE3_Rkeys (i.e., DE3 with Rkeys), etc.

Furthermore, in the experimental comparisons we included a novel slight modification of the original DE algorithm. This modification concerns the acceptance rule and is given by the relation,

$$x_{i,k+1}^{j} = \begin{cases} r^{j} & \text{if } R^{j} \le \rho \\ x_{i,k+1}^{j} & \text{otherwise} \end{cases}, \quad \forall j = \{1, 2, \ldots, D\} \tag{11}$$

where, $x_{i,k+1}^{j}$the j-th component of the updated target vector $x_{i,k+1}$, $i = 1, 2, \ldots N$. $R^{j} \neq r^{j}$are random numbers drawn uniformly within the range (0,1) for each$j$, and ρ a small (user-defined) fixed probability. After experimentation we found $\rho = 0.1$ to be a good choice for our study. The idea with Eq. (11) is to enforce more the exploration ability of the algorithm by replacing with probability ρ the components of the updated target vector (created by Eq. (4)) with random numbers. Numerical results showed that the use of this simple change improves significantly the performance of the original DE algorithm, especially when used with DE3 heuristic. Therefore, we decided to adopt DE3 with the proposed modification and refer to the new heuristic as mDE3.

All the DE variants were coded in Pascal programming language and run on a Pentium IV 3.2 GHz PC. The evaluations were performed over public benchmark test problems with known lower (or upper) bounds. To quantify the generated solutions two performance criteria were used: (a) the average percentage offset from the global optimum solution ($\Delta_{\text{avg}}\%$), and (b) the average percentage solution effort ($\text{E}_{\text{avg}}\%$). The two criteria are given below:

$$\Delta_{\text{avg}}\% = \frac{\text{Cost}_{DE} - C^{*}}{C^{*}} \times 100 \tag{12}$$

Where, Cost_{DE}, is the cost of the best solution achieved by a heuristic for a specific benchmark test instance, and C^{*} is the corresponding exact optimal (or near-optimal) solution reported in the literature for this test instance. Thus, for the case of FSSP, Cost_{DE} corresponds to the makespan (Eq. (8)), while for TWTP and CDDSP, Cost_{DE} corresponds to the summations given by Eqs. (9) and (10), respectively. The second criterion is,

$$E_{\mathrm{avg}}\% = \frac{k_{\mathrm{opt}}}{\mathrm{MAXI}} \times 100 \tag{13}$$

In Equation (13), k_{opt} is the iteration number at which the best solution for a specific test instance has been achieved, and $\mathrm{MAXI} = 500 \times n$, is the maximum number of iterations an algorithm run; n is the number of the jobs to be scheduled (i.e., the size of the COP). It is underlined, that all the tested algorithms were left running for the same MAXI number of iterations.

6.2 Choice of the control parameters' settings

The three control parameters (population size $N \geq 4$, crossover rate $C_R \in [0, 1]$, and mutation-scale factor $F \in (0, 2)$) included in the components of the DE algorithm, and the many possible choices make the determination of the perfect settings a very difficult task, almost impossible. Much investigation on the selection of the appropriate settings of these parameters was undertaken in preliminary tests. In this sub-section we give a detailed description of the experimental design methodology used to determine these settings.

In particular, N was defined to vary with the size of the COP, i.e., $N = n$ (= number of the jobs to be scheduled). Moreover, N remains fixed during the algorithms' execution. For C_R and F two different control schemes were used: a static scheme where the values of the parameters are kept fixed for the whole life of the evolutionary process, and a dynamic scheme where parameters' values are varying through DE run.

More specific, in the static scheme, C_R was defined to take values within the following discrete range $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. In the dynamic scheme, C_R was defined to be adapted per iteration by the relation,

$$\text{if}\,\mathrm{Cost}_{\min} \geq T \times \mathrm{Cost}_{\mathrm{avg}} \quad \text{then}\, C_R = C_{R\_\mathrm{init}}\, \text{else}\, C_R = \Theta \times C_R \tag{14}$$

This scheme gives to C_R a high value at the beginning of the run and decreases this rate slowly by the diversity of the population. C_R is initially defined equal to $C_{R\_\mathrm{init}} = 0.8$, and decreased in each new iteration by a factor $\Theta = 0.95$ using the linear relation $C_R = \Theta \times C_R$. If the minimum generated cost ($\mathrm{Cost}_{\min}$) of the current population becomes almost the same to the average population cost ($\mathrm{Cost}_{\mathrm{avg}}$), then a very small diversity is encountered in the population and thus the crossover rate is reset to the initial value C_{R_init}. Moreover, in the case where C_R becomes very small (<0.05), it is again reset to C_{R_init}. $T(= 0.97)$ is a user-defined threshold.

For parameter F, the following static control scheme was used: firstly $F \in [0.5, 0.7, 0.9]$, and secondly F is estimated by a rule proposed by Zaharie (2002). This rule considers the values of the control parameters satisfying the equation $2F^2 - 2/N + C_R/N = 0$ to be critical for the convergence ability of DE. Zaharie proposed this rule after an analysis of the influence of the mutation and crossover operators on the expected population variance of DE when applied for optimization over continuous spaces. In the case of the dynamic control scheme, we experimented with two different strategies recently proposed by Kaelo and Ali (2006) and Ali and Törn (2004), respectively. According to the first strategy, for each mutated point, F is randomly chosen within the range $[-1, -0.4] \cup [0.4, 1]$ (Kaelo and Ali, 2006). While according to the second strategy, F should take large values in the early stages and small values at the later stages of the DE algorithm. This strategy (Ali and Törn, 2004), is

Table 2 Choosing the correct settings for the parameters C_R and F. Average %offset over characteristic benchmarks for the three COPs: (a) FSSP, (b) TWTP, and (c) CDDSP

	F					
C_R	0.50	0.70	0.90	Zaharie's rule	Ali & Törn scheme	Kaelo & Ali scheme
				(a) FSSP		
0.1	2.41	1.96	1.90	2.03	1.83	1.83
0.3	2.29	2.47	1.90	2.03	2.31	2.31
0.5	2.72	2.63	2.80	2.39	2.32	2.32
0.7	2.33	2.87	2.75	2.38	2.92	2.92
0.9	2.54	2.57	2.92	2.40	2.81	2.81
$C_R = \Theta \times C_R$	1.69	1.37	2.12	**1.22**	1.90	2.01
				(b) TWTP		
0.1	0.93	0.92	0.88	0.78	0.93	0.89
0.3	2.09	3.63	2.77	4.67	2.18	3.94
0.5	7.85	6.58	16.59	5.80	6.23	6.54
0.7	19.93	10.10	8.03	6.47	23.29	9.19
0.9	13.47	16.10	8.64	17.18	18.52	7.57
$C_R = \Theta \times C_R$	0.90	0.88	0.87	0.86	0.86	**0.69**
				(c) CDDSP		
0.1	−3.82	−3.64	−3.57	−3.84	−3.82	−3.75
0.3	−3.55	−3.53	**−3.84**	−3.75	−3.55	−3.60
0.5	**−3.84**	−3.82	−3.77	−3.39	**−3.84**	−3.77
0.7	−3.83	−3.64	−3.82	**−3.84**	−3.83	−3.83
0.9	−3.64	−3.58	−3.67	−3.60	−3.64	−3.68
$C_R = \Theta \times C_R$	−3.82	−3.57	−3.73	−3.82	−3.74	**−3.84**

implemented by the relation,

$$F = \begin{cases} \max\left(0, 1 - \left|\frac{f_{\max}}{f_{\min}}\right|\right) & \text{if} \left|\frac{f_{\max}}{f_{\min}}\right| < 1 \\ \max\left(0, 1 - \left|\frac{f_{\min}}{f_{\max}}\right|\right) & \text{otherwise} \end{cases} \tag{15}$$

where, $f_{\max}$ and $f_{\min}$ are the maximum and minimum function values in S, respectively. F takes its values within the range [0,1].

Since we described the experimental design methodology for the determination of C_R and F, we can now study the effect of these control schemes on DE. To that purpose, we selected three characteristic sets of test instances taken from a well-known, public benchmarks library (at http://mscmga.ms.ic.ac.uk/). These instances are: the 10 instances of the Taillard's 20-jobs, 10-machines FSSP, the first 20 instances contained in the 40-jobs TWTP, and the 10 instances of the 20-jobs CDDSP. Each one of the tested DE variants was run 30 times over the selected characteristic test instances and the results obtained were averaged. Table 2 displays the generated $\Delta_{\text{avg}}\%$ using the various control schemes. Note that, only the results due to the best DE heuristic are presented in the table, which is mDE3_SR. A quite similar effect of C_R and F on the performance of the other DE variants was also investigated.

As one can observe from Table 2(a), the best result for FSSP ($\Delta_{\text{avg}}\% = 1.22$) was obtained using the proposed adapted C_R scheme (Equation (14)) and the Zaharie's rule for the estimation of F. For TWTP (Table 2(b)) best results were obtained using the adapted C_R scheme and F estimated by Kaelo's and Ali's strategy. While, for CDDSP (Table 2(c)), the previous combination, i.e. adaptable C_R and F estimated by the strategy of Kaelo and Ali (2006) was again one of the most effective choices for DE. Hence, the following control schemes were finally adopted for the experimental evaluations: adaptable C_R for all COPs, F estimated by Zaharie's rule for FSSP, and F estimated by Kaelo's and Ali's strategy for both TWTP and CDDSP.

6.3 The FSSP benchmark tests

The DE heuristics were first tested over the well-known Taillard's benchmarks FSSPs (Taillard, 1993). Taillard's problems have been found to be very difficult, in the sense that the best solutions found till now are through the use of a very lengthy Tabu-search heuristic (Taillard, 1990). These benchmarks range from small size instances with 20 jobs on 5 machines to large size instances with 500 jobs on 20 machines. There are ten instances for each size problem, with each instance having a different lower bound. To get the average performance of the heuristics, ten runs on each test instance were performed and the solution quality was averaged.

Table 3 illustrates the results obtained by the DE heuristics over these benchmarks with up to 200 jobs on 20 machines. The results correspond to the mean percentage offset from the existing upper bounds ($\Delta_{\text{avg}}\%$), averaged over the ten instances within each category of problems, and the mean % effort ($E_{\text{avg}}\%$) spent by the heuristics until the convergence. The best solutions found in each problem's category are illustrated in boldfaced. For instance, for the case of the 20×5 (stands for 20 jobs on 5 machines) category of problems, the best result ($\Delta_{\text{avg}}\% = 0.93$) was achieved by the proposed heuristic mDE3_SR, while the second best performance ($\Delta_{\text{avg}}\% = 1.24$) was achieved by DE2_SR. From Table 3 one can clearly observe that the use of the proposed solutions' encoding scheme (SR) within DE results to a substantially higher performance than that obtained using Rkeys. DE with SR managed to generate solutions of much higher quality (schedules with shorter makespans), in all the experiments. Note that, for DE with Rkeys we present only the results obtained by the best DE variant, which is DE3. The two other schemes (DE1 and DE2) with Rkeys performed even worse.

More specifically, mDE3_SR outperformed the other DE heuristics in all categories of problems obtaining solutions with an average percentage offset equal to 2.13 (see the last

Table 3 Comparative results between the five heuristics over the Taillard's FSSP benchmarks

Problem	DE1_SR		DE2_SR		DE3_SR		DE3_RKeys		mDE3_SR	
$n \times m$	$\Delta_{\text{avg}}\%$	$E_{\text{avg}}\%$	$\Delta_{\text{avg}}\%$	$E_{\text{avg}}\%$	$\Delta_{\text{avg}}\%$	$E_{\text{avg}}\%$	$\Delta_{\text{avg}}\%$	$E_{\text{avg}}\%$	$\Delta_{\text{avg}}\%$	$E_{\text{avg}}\%$
20 × 5	2.53	22.12	1.24	10.87	1.33	3.93	9.32	7.70	**0.93**	30.62
20 × 10	2.50	27.88	1.82	21.63	1.50	21.19	16.11	2.97	**1.22**	28.36
20 × 20	3.05	12.30	1.68	7.03	1.37	13.82	11.23	2.83	**1.04**	26.17
50 × 5	1.22	13.10	1.39	4.86	1.42	10.04	7.07	4.80	**0.42**	47.81
50 × 10	5.21	46.22	2.31	64.15	3.02	41.35	17.11	5.14	**2.83**	46.61
50 × 20	5.88	40.17	4.75	64.36	4.12	60.42	19.78	18.84	**3.79**	53.93
100 × 5	1.37	7.92	1.41	9.24	1.29	9.06	6.20	1.49	**0.34**	36.66
100 × 10	4.23	22.13	2.44	51.30	2.48	51.83	16.09	9.06	**2.16**	45.03
100 × 20	5.21	6.15	4.72	70.28	4.65	47.72	15.92	4.19	**4.03**	68.41
200 × 10	3.42	63.39	2.82	22.24	2.77	15.36	20.56	2.15	**2.06**	73.87
200 × 20	5.41	26.38	5.71	31.97	5.26	23.14	23.65	3.93	**4.66**	67.50
Average	3.64	26.16	2.75	32.54	2.66	27.08	14.82	5.74	**2.13**	47.72

line of Table 3). Very near to this performance are firstly, the results obtained by DE3_SR (a mean $\Delta_{\text{avg}}\% = 2.66$), and secondly the results generated by DE2_SR (a mean $\Delta_{\text{avg}}\% = 2.75$). DE1_SR gave results of even lower quality with a mean $\Delta_{\text{avg}}\% = 3.64$. The use of Rkeys encoding failed to produce schedules of low makespan. In all the experiments, the results obtained using Rkeys were of very poor quality with an average offset from optimum approx. equal to 14.8%. A result indicating that this encoding scheme is unsuitable for use within DE to address FSSPs.

Examining the computational effort, DE1_SR seems to be the fastest with a mean effort approx. equal to 26%. Near to this bound comes the effort spent by DE3_SR ($\text{E}_{\text{avg}}\% \approx 27\%$). The low average effort ($\approx$6%) spent by DE3_Rkeys, is a characteristic caused by the premature convergence of this algorithm. Actually, due to its poor effectiveness, this algorithm very soon stacks to a (bad) local minima solution.

6.4 The TWTP benchmark tests

For the case of TWTP we used the famous benchmarks set available via the public ORLIB library (web-location: http://mscmga.ms.ic.ac.uk/). This set comprised instances with 40, 50 and 100 jobs. For the 40 and 50 jobs the optimal solutions are known, while for the 100 job instances only the best so far (near-optimal) solutions are known. Each category of problems contains 125 test instances. Each test instance includes for each job j ($j = 1,2, \ldots, n$) a processing timep_j, a weight w_j, and a due date d_j (as analyzed in Section 3.2). The results obtained by the heuristics over the above benchmarks are summarized in Table 4. To quantify the quality of the generated solutions the following three performance indices were used: $\Delta_{\text{avg}}\%$ (Equation (12)$_,$ the number of exact optimal solutions found (n_{opt}), and $E_{\text{avg}}\%$ (Equation (13).

One can easily see from Table 4, that the performance of the DE variants with SR, are by far superior from that of DE with Rkeys. Rkeys is too weak for use within DE to address TWTP. Consequently, our observation and analysis is limited on the relative performance of the DE heuristics with SR. These heuristics managed to generate solutions with a mean offset from optimum less than 2% in the case of the 40-jobs TWTP, less than 3% in the case of 50-jobs TWTP, and less than 2.5% in the case of the 100-jobs TWTP. mDE3_SR heuristic became again the champion producing solutions with lower cost than the others. In

Table 4 Comparative results between the DE heuristics over the TWTP benchmarks. Test instances with (a) 40-jobs. (b) 50-jobs, and (c) 100-jobs

	DE1_SR	DE2_SR	DE3_SR	DE3_RKeys	mDE3_SR
			(a) 40 jobs TWTP		
Δ_{avg} %	1.90	1.52	1.09	16.20	**0.38**
n_{opt}	26	27	23	–	**88**
E_{avg} %	34.9	26.3	53.07	41.09	43.17
			(b) 50 jobs TWTP		
Δ_{avg} %	2.89	1.90	1.60	23.05	**1.12**
n_{opt}	28	26	25	–	**44**
E_{avg} %	31.9	59.81	44.72	53.90	61.89
			(c) 100 jobs TWTP		
Δ_{avg} %	2.26	2.33	2.13	30.16	**1.83**
n_{opt}	16	27	32	–	**37**
E_{avg} %	29.04	59.11	46.32	42.84	43.91

Table 5 Seeding the initial population with a solution built by constructive rules. $\Delta_{\text{avg}}\%$ obtained by mDE3_SR in combination with constructive rules over the 100-jobs TWTP instances

EDD	MDD	AU	No seed
1.34	2.23	2.28	1.83

particular, mDE3_SR produced solutions with $\Delta_{\text{avg}}\% = 0.38\%$ in the case of 40-jobs (Table 4(a)), $\Delta_{\text{avg}}\% = 1.12\%$ in 50-jobs (Table 4(b)), and $\Delta_{\text{avg}}\% = 1.83\%$ in 100-jobs TWTP (Table 4(c)). Moreover, mDE3_SR achieved the exact optimum solution in much more instances than the remaining heuristics: particularly, in 88 out the total 125 of the 40-jobs, in 44 instances of the 50-jobs, and in 37 out the total 125 instances of the 100-jobs TWTP. The second best performance was again achieved by DE3_SR heuristic.

Furthermore, the effect of seeding the initial population of the DE algorithm with a single solution generated by a constructive TWT rule was examined. The goal here is to investigate whether or not, seeding the initial population of DE results to better final solutions. Three of the most referred in the related literature quick constructive rules were included in the comparisons namely, Earliest Due Date (EDD), Apparent Urgency (AU), and Modified Due Date (MDD) (Potts and Wassenhove 1991, Crauwels et al., 1998).

- EDD technique sorts the jobs by ascending order of their due dates d_j $(j = 1, 2, \ldots, n)$.
- AU sequences the jobs based on the apparent urgency, which is defined as $AU_j = (w_j/p_j) \cdot \exp(-\max(0, d_j - C_j - p_j)/\mu \cdot \bar{P})$ $(\forall j = 1, 2, \ldots, n)$. Where, p_j the processing time of the *j*th job, w_j a weight denoting the relative priority of job *j*, and d_j the due date for job *j*. C_j is the completion time of job *j* in the generated job sequence μ is a user defined parameter (called the 'look-ahead' parameter) set according to the tightness of the due dates (here, $\mu = 2$). $\bar{P}$ is the average processing time.
- MDD puts the jobs in non-decreasing order of the modified due dates, given by the relation $mdd_j = \{C + p_j, d_j\}$. Where *C* is the sum of the processing times of the already sequenced jobs.

Table 5 displays Δ_{avg} % obtained by the best heuristic mDE3_SR in combination with each one of the three constructive heuristics mentioned above. The results concern the 100-jobs TWTP benchmarks. As one can see from Table 5, seeding the initial population with a solution generated by EDD rule results to the highest performance. This combination (mDE3_SR + EDD) managed to generate solutions with Δ_{avg} % = 1.34%. While the use of MDD and AU rules performed worse, resulting to final solutions of lower quality.

6.5 The CDDSP benchmark tests

We now compare the performance of the tested DE heuristics on CDDSP. The comparisons were carried out on a set of public benchmarks recently proposed by Biskup and Feldmann (2001). These benchmarks include 280 test instances ranging from small size instances with 10 jobs to large size instances with 1000 jobs. There are ten instances for each size problem. A parameter h (= 0.2, 0.4, 0.6, 0.8) was used by (Biskup and Feldmann, 2001) to produce test instances with more or less restrictive common due dates. Test instances generated with $h = 0.2$ are harder to be solved than those generated with greater values of h. The following discussion concerns the application of the algorithms on benchmarks instances with up to 200 jobs generated with $h = 0.2$. For each test instance we are given the number of the jobs

Table 6 Comparative results over public benchmark problems for CDDSP

Problem	DE1_SR		DE2_SR		DE3_SR		DE3_RKeys		mDE3_SR	
n	$\Delta_{avg}\%$	$E_{avg}\%$	$\Delta_{avg}\%$	$E_{avg}\%$	$\Delta_{avg}\%$	$E_{avg}\%$	$\Delta_{avg}\%$	$E_{avg}\%$	$\Delta_{avg}\%$	$E_{avg}\%$
10	4.73	17.51	**0.00**	3.17	**0.00**	6.02	1.21	26.25	**0.00**	11.76
20	3.97	2.09	−0.68	19.76	−0.51	12.77	12.70	42.33	**−3.84**	7.74
50	6.00	19.41	−0.30	17.56	−0.24	10.81	30.12	31.73	**−5.57**	57.03
100	6.23	6.07	−4.80	33.05	−4.37	21.50	44.02	28.98	**−5.93**	97.62
200	8.30	11.35	−0.09	29.40	−0.11	39.98	53.10	28.30	**−5.04**	92.98
Average	5.85	11.29	−1.17	20.59	−1.05	18.22	28.23	31.52	**−4.08**	53.43

to be scheduled (n), a due date (d) common for all jobs, and for each job j ($j = 1, 2, \ldots, n$) its processing time p_j, and two penalty coefficients α_j, β_j for the earliness and the tardiness, respectively.

Optimal solutions for the examinant benchmarks exist only for the instances with 10 jobs and have been achieved using an integer programming formulation with LINDO software. Table 6 illustrates the comparative results (averaged over the 10 instances of each problem's category) obtained by the five DE variants over these benchmarks. The best results obtained are indicated in the table in boldfaced. The negative sign in some results denotes that the corresponding heuristic managed to improve the existing upper bound producing solutions of lower costs.

As one can see from Table 6, the performance of DE2_SR, DE3_SR, and mDE3_SR are of high quality, with the latter being the best of all, producing solutions with a mean $\Delta_{avg}\%$ over the five categories equal to −4.08%. This means that mDE3_SR improved the existing upper bounds of Biskup's and Feldmann's benchmarks by approx. 4% in average. For DE2_SR and DE3_SR an average improvement to the upper bounds greater than 1% is reported. Furthermore, it is worth reporting that, these three heuristics reached the exact optimum solution in all the test instances with 10 jobs, and improved the existing bounds for all the test instances of the remaining CDDSPs. The highest improvement concerns the

Table 7 Running times in CPU (seconds) on a Pentium 3.2 GHz PC

FSSP	Problem size (n)				
Method	20	50	100	200	
DE + SR	1.96	56.45	694.16	14,655.05	
DE + RKeys	0.91	46.39	385.94	3,687.10	
TWTP	Problem size (n)				
Method	40	50	100		
DE + SR	28.94	58.40	631.06		
DE + RKeys	0.00	0.16	308.50		
CDDSP	Problem size (n)				
Method	10	20	50	100	200
DE + SR	0.90	3.37	56.16	835.63	11,027.52
DE + RKeys	0.81	3.09	37.66	311.40	4,992.39

100-jobs instances. Here, the reported averaged improvements are approx. 4.8%, 4.4%, and 5.9% for DE2_SR, DE3_SR, and mDE3_SR, respectively. Examining the % effort, we can see that, mDE3_SR spent in average more time than the other DE variants until the convergence. DE with Rkeys produced acceptable solutions only for the case of 10-jobs instances (Δ_{avg}% = 1.21%). A rather poor performance is also achieved by DE1_SR. An event not observed in the case of FSSP and TWTP.

Finally, for the completeness of this paper, we report in Table 7 the average running times (in CPU seconds) needed by DE using the two encodings (SR, Rkeys). The times are depended on the type and the size of the COP. As it is obvious, the use of Rkeys encoding within DE results to a much faster algorithm than that of using SR.

7 Conclusion

This paper investigates the application of the differential evolution (DE) algorithm on the solution of three classic NP-hard scheduling problems: the multiple machine flow-shop scheduling problem, the single machine total weighted tardiness problem, and the single machine common due date scheduling problem. DE is a recently developed heuristic that has empirically proven to be very robust for global optimization over continuous spaces. By incorporating in DE a new representation scheme for solution encoding, it was experimentally shown that DE is also quite robust for discrete optimization problems. Moreover, a very simple modification of the original DE algorithm was introduced and tested on public available benchmarks for the three scheduling problems. The experimental comparisons showed that the combination of the proposed encoding scheme with the new DE variant outperforms the original DE algorithm for discrete optimization. Furthermore, the use of the proposed encoding scheme within DE was found substantially superior to the use of the famous random-keys technique. Random-keys encoding was found to be not suitable for use with DE for discrete optimization.

We do not claim that the DE algorithm is more robust than existing effective heuristics for any of the three problems presented. Instead, our main purpose was to show how DE can address a wide range of combinatorial problems with discrete decision variables and to measure its relative performance. To the best of our knowledge these are the first reported results concerning the application of DE over public benchmarks with known optimal solutions for the referred scheduling problems. Future work will investigate the application of DE on other more complex combinatorial optimization problems such as the job-shop scheduling problem. Moreover, the results obtained over the benchmarks of the common due date scheduling problem were very encouraged. On going research is concentrated on the development of a more robust version of DE with the hope to improve more the existing upper bounds for the Biskup's and Feldmann's (2001) benchmarks. Hybridizing DE with suitable local search techniques is a promising area of research to start with.

Acknowledgments This work is integrated in the Innovative Production Machines and Systems (I*PROMS) Network of Excellence.

References

Abdul-Razaq, T.S., C.N. Potts, and L.N. Van Wassenhove. (1990). "A Survey of Algorithms for the Single Machine Total Weight Tardiness Scheduling Problem." *Discrete Applied Mathematics* 26, 235–253.

Ali, M.M. and A. Törn. (2004). "Population Set-based Global Optimization Algorithms: Some Modifications and Numerical Studies." *Computers & Operations Research* 31, 1703–1725.

Ali, M.M., C. Khompatraporn, and Z.B. Zabinsky. (2005). "A Numerical Evaluation of Several Stochastic Algorithms on Selected Continues Global Optimization Test Problems." *Journal of Global Optimization* 31, 635–672.

Baker, K. and G. Scudder. (1990). "Sequencing with Earliness and Tardiness Penalties: A Review." *Operations Research* 38, 22–36.

Bean, J. (1994). "Genetics and Random Keys for Sequencing and Optimization." *ORSA Journal on Computing* 6(2), 154–160.

Biskup, D., and M. Feldmann. (2001). "Benchmarks for Scheduling on a Single Machine Against Restrictive and Unrestrictive Common Due Dates." *Computers and Operations Research* 28, 787–801.

Cheng, S.-L. and C. Hwang. (2001). "Optimal Approximation of Linear Systems by a Differential Evolution Algorithm." *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 31(6), 698–707.

Crauwels, H.A.J., C.N. Potts, and L.N. Van Wassenhove. (1998). "Local Search Heuristics for the Single Machine Total Weighted Tardiness Scheduling Problem." *INFORMS Journal on Computing* 10(3), 341–350.

Kaelo, P. and M.M. Ali. (2006). "A numerical study of some modified differential evolution algorithms." *European Journal of Operational Research* 171, 674–692.

Lin, Y.-C., K.S. Hwang, and F.-S. Wang. (2004). "A Mixed-Coding Scheme of Evolutionary Algorithms to Solve Mixed-Integer Nonlinear Programming Problems." *Computers & Mathematics with Applications* 47(8–9), 1295–1307.

Nearchou, A.C. (2006). "Meta-Heuristics from Nature for the Loop Layout Design Problem." *International Journal of Production Economics* 101/2, 312–328.

Onwubolu, G.C. (2004). "Optimizing CNC Drilling Machine Operations: Traveling Salesman Problem-Differential Evolution Approach." In Onwubolu, G. C. and Babu, B. V. (eds.), *New Optimization Techniques in Engineering*, Springer-Verlag, Heidelberg, Germany, pp. 537–565.

Potts, C.N. and L.N. Van Wassenhove. (1991). "Single Machine Tardiness Sequencing Heuristics." *IEE Transactions* 23, 346–354.

Rinnooy, A.H.G. Kan (1976). *Machine Scheduling Problems: Classification, Complexity and Computations*. Martinus Nijhoff, The Hague.

Storn, R. and K. Price. (1997). "Differential Evolution—A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces." *Journal Global Optimization* 11, 241–354.

Sun, J., Q. Zhang, and E.P.K. Tsang. (2005). "DE/EDA: A New Evolutionary Algorithm for Global Optimization." *Information Sciences* 169(3–4), 249–262.

Taillard, E. (1990). "Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem". *European Journal of Operational Research* 47, 65–74.

Taillard, E. (1993). "Benchmarks for Basic Scheduling Problems." *European Journal of Operational Research* 64, 278–285.

Zaharie, D. (2002). "Critical Values for the Control Parameters of Differential Evolution Algorithms." In: Matouek, Radek and Omera, Pavel (eds.). *Proc. of MENDEL 2002, 8th Int. Mendel Conference on Soft Computing*, Brno, Czech Republic, pp. 62–67.