Flow Shop Scheduling Using a General Approach for Differential Evolution

Frederico Gadelha Guimarães, Rodrigo César Pedrosa Silva,
Ricardo Sérgio Prado, Oriane Magela Neto, and Donald David Davendra

Abstract. This chapter presents a general framework of Differential Evolution algorithm for combinatorial optimization problems. We define the differences between a given pair of solutions in the differential mutation as a set of elementary movements in the discrete search space. In this way, the search mechanism and self-adaptive behavior of the differential evolution is preserved and generalized to combinatorial problems. These ideas are then applied to n-job m-machine flow shop scheduling in order to illustrate its application in an important problem in combinatorial optimization. The method was applied to the 120 Taillard instances of the permutation flow shop scheduling problem, and compared against the results obtained by other metaheuristic algorithms in the literature. Although relying only on the differential mutation and the local search performed on the best individual, dDE ranks fairly well against more sophisticated metaheuristics. The results are promising and illustrate the applicability of the proposed approach for combinatorial optimization using differential evolution.

Frederico Gadelha Guimarães · Rodrigo César Pedrosa Silva · Oriane Magela Neto
Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais,
Belo Horizonte, Brazil
e-mail: fredericoguimaraes@ufmg.br,
rcpsilva@gmail.com,
oriane@dee.ufmg.br

Ricardo Sérgio Prado
Instituto Federal Minas Gerais, Ouro Preto, Brazil
e-mail: ricardo.prado@ifmg.edu.br

Donald David Davendra
Department of Computer Science, Faculty of Electrical Engineering and Computer Science, VB-Technical University of Ostrava, Czech Republic
e-mail: donald.davendra@vsb.cz

1 Introduction

Scheduling is a very important task in production and manufacturing systems. In addition to representing a practical problem faced on a daily basis in industry, it is also an important theoretical challenge in computer science, since it is an NP-hard combinatorial optimization problem [4, 12, 11]. The n-job, m-machine flow shop scheduling problem (FSSP) is one of the most general job scheduling problems, representing nearly a quarter of manufacturing systems, assembly lines and information service facilities nowadays [26].

Given its importance in industry and also from a scientific point of view, FSSP has been studied by many reseachers and approached with a myriad of optimization techniques, ranging from exact methods such as branch-and-bound to heuristic and metaheuristics methods such as genetic algorithms, tabu search, differential evolution, simulated annealing and others. For a comprehensive survey on FSSP, see [16, 3].

Metaheuristics have been extensively applied to solve practical instances of difficult optimization problems. These methods can reach high quality solutions in an effective and efficient way, i.e. within an acceptable computational budget. Evolutionary algorithms, and genetic algorithms in particular, have been attracted a lot of research in scheduling problems, maybe due to their parallel exploration of the search space, and their potential to be implemented in parallel and be associated with other heuristics such as local search methods [9]. For instance, [3] mentions the genetic algorithm as the most used metaheuristic for scheduling problems.

Among the evolutionary techniques that can be applied to optimization, the differential evolution (DE) algorithm has been showing a very good performance in dealing with hard and complex optimization, being a powerful method for single objective optimization [28, 27, 18], constrained problems [14, 13], and more recently multi-objective problems as well, see [1, 17, 40, 5] just to cite a few. There have been some research on the use of DE also for combinatorial problems such as permutation job shop and flow shop problems [30, 25, 24, 22, 21].

The self-adaptive nature of the search mechanism of differential evolution is perhaps responsible for the success of the method: in the first generations, when the population is more diverse, the differences between pairs of individuals are varied and diverse, leading to exploratory perturbations. As the population converges, these differences reduce gradually, and similar values begin to appear at the same positions (or variables). The distribution of the population in the search space begins to correlate with the problem landscape structure. The differences between pairs of individuals in the population are used to bias the mutation, therefore identifying "promising" perturbations. The diversity and the magnitude of these perturbations tend to reduce during convergence, generally adapting itself to the characteristics of the problem landscape. This behavior is easy to see in continuous domains and has been shown elsewhere [29].

However, when dealing with combinatorial optimization problems, this behavior is not that straightforward. When we resort to the concept of elementary moves in the search space of combinatorial problems, which is the basis for any local search procedure, than this interpretation of the differential evolution becomes easier to grasp. Suppose that the algorithm is able to define the differences between a given pair of solutions as a set of elementary movements (or basic "jumps") in the discrete search space (the specific details about how this is done is not important for the discussion here, this is explained further in this chapter). Assuming that this is available, we can generalize the algorithm behavior to combinatorial problems.

Again, during the initial generations, when the population is more diverse, the set of differences tend to have a high number of diverse movements in them affecting many variables of the solutions. In this initial phase, the differential mutation have an exploratory role in the algorithm, similarly to what happens in real valued problems. As the population converges, some individuals with the same values in some positions begin to appear, reducing the differences between the solutions used to build the differential mutation, consequently reducing the magnitude of the mutation. The set of differences identifies elements that vary in two given solutions, not altering the common values identified so far, and capturing the elementary movements related to undefined positions of the good schema identified so far. When these differences are applied to a base solution, in the form of a sequence of movements, it modifies those positions related to undefined positions identified in pairs of solutions of the population. Therefore, the elements that vary in pairs of solutions in the population, i.e. the differences between these solutions, are used to build and to bias the perturbations applied to the base solution.

The size and diversity of the differences automatically decrease as the population converges, presenting the same self-adaptive characteristic that makes DE such a successful optimizer in continuous domains. The differential mutation operator automatically changes its role from a global search operator to a local search one, fine-tuning the high quality solutions in the population. As long a mechanism for identifying differences in the context of combinatorial optimization is adequately defined, the search mechanism of the differential evolution is preserved and the interpretation of its behavior is still valid. In this chapter, we discuss ways to build these differences, proposing a general framework for discrete differential evolution. There have been some approaches in recent years for applying DE in discrete domains [15, 21], some of them are reviewed in this chapter and contrasted with the proposed approach. With the proposed approach to get the differences in a discrete search space, like the ones originated in permutation-based combinatorial problems, we preserve the search mechanism of DE to combinatorial problems. These ideas are then applied to flow shop scheduling in order to illustrate its application in an important problem in combinatorial optimization.

2 Differential Evolution

DE can be classified as a member of the broader class of population-based evolutionary algorithms, thus inheriting much of the terminology and jargon of this class of methods, though its main search mechanism, the differential mutation operation, has no basis or inspiration on any biological process. Nevertheless, DE structure follows the overall structure of evolutionary algorithms, with the main operations of *selection*, *crossover*, and *mutation*, applied to a population of candidate solutions of a given optimization problem [27, 28, 29]. The selection for reproduction in DE is similar to the one used in basic evolutionary programming, in which each individual produces one and only one offspring at every generation. The offspring is generated by performing crossover between the current individual and a mutant. The distinguishing feature in DE is that these mutants are produced by applying a perturbation built with one or more difference vectors to a base solution. The way the perturbation is built and the way the base vector is defined determines one of many variants of the basic DE. Finally, survival selection in DE is a one-to-one greedy selection, in which the offspring competes against the current solution for a place in the population of the next generation.

2.1 Overview

As many population-based metaheuristics, DE starts with a population of candidate solutions randomly generated within the domain region of the problem. The idea is to have as high a diversity as possible in the first generation. Over continuous domains the domain region is usually described as:

$$\mathcal{X} = \left\{\mathbf{x} \in \mathbb{R}^n : x_k^{\min} \leq x_k \leq x_k^{\max}, k = 1, \ldots, N\right\} \tag{1}$$

where $x_k^{\min}$ and $x_k^{\max}$ are respectively the low and upper limits of each variable.

We adopt the notation $x_{t,i,j}$ for making reference to a given variable, such that $t = 1, \ldots, G$ represents the generation counter; $i = 1, \ldots, P$ represents the index of the individual in the population; and $j = 1, \ldots, N$ represents the variable index. A given individual is represented by $\mathbf{x}_{t,i}$.

New individuals are generated by using the differential mutation. For each $\mathbf{x}_{t,i}$ in the population a corresponding mutant solution $\mathbf{v}_{t,i}$ is generated. The mutation is based on the differences between pairs of individuals randomly chosen from the current population. These differential vectors are multiplied by a constant and added to another point, called the base vector (or base solution), leading to the so-called mutant vector:

$$\mathbf{v}_{t,i} = \mathbf{x}_{base} + \sum_{k=1}^{d} F_k \Delta\mathbf{x}_{t,k} = \mathbf{x}_{base} + \sum_{k=1}^{d} F_k \left(\mathbf{x}_{t,r_k} - \mathbf{x}_{t,r_{k+d}}\right) \tag{2}$$

where r_k and r_{k+d} represent random integers in the interval $[1, P]$.

For instance, using only one difference vector in (2), we get:

$$\mathbf{v}_{t,i} = \mathbf{x}_{base} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right) \tag{3}$$

whereas with $d = 3$, we have:

$$\mathbf{v}_{t,i} = \mathbf{x}_{base} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_4}\right) + F\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_5}\right) + F\left(\mathbf{x}_{t,r_3} - \mathbf{x}_{t,r_6}\right) \tag{4}$$

There are a number of different ways of defining the base solution, which are listed below:

- The most common way is to use another random solution from the population. In this way, every individual has a probability $1/P$ of generating a mutant. In this case, $\mathbf{x}_{base} = \mathbf{x}_{t,r_0}$, with r_0 representing a random index in $[1, P]$.
- Another common approach is to use $\mathbf{x}_{base} = \mathbf{x}_{t,best}$, i.e., the base solution is simply the best solution in the population. Using the best solution as the base vector in all differential mutation operations increases the selective pressure in reproduction, contrasting to the lack of selective pressure of using a random base solution. This strategy presents faster convergence in general, but an increased probability of premature convergence in some problems.
- The mean of the current spatial distribution of the population is also another option for the base solution. In this way we have $\mathbf{x}_{base} = \mathbf{x}_{mean}$, such that:

$$\mathbf{x}_{mean} = \frac{1}{P}\sum_{l=1}^{P} \mathbf{x}_{t,l}$$

- In order to reduce the selective pressure of using $\mathbf{x}_{t,best}$ or $\mathbf{x}_{mean}$, one can use as base vector a random point between $\mathbf{x}_{t,i}$ and $\mathbf{x}_{t,best}$:

$$\mathbf{x}_{base} = \mathbf{x}_{t,i} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,i}\right)$$

  or even a random point between $\mathbf{x}_{t,r_0}$ and $\mathbf{x}_{t,best}$:

$$\mathbf{x}_{base} = \mathbf{x}_{t,r_0} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,r_0}\right)$$

  This form of selection of the base vector bias the generation of mutants towards the best solution.

A trial vector $\mathbf{u}_{t,i}$ is produced through recombination of $\mathbf{x}_{t,i}$ and $\mathbf{v}_{t,i}$. In the basic DE algorithm, the discrete recombination with probability CR is used. In this way, F and CR represent the control parameters of the algorithm. Other recombination operators can be used as well.

This trial vector $\mathbf{u}_{t,i}$ competes against the current solution $\mathbf{x}_{t,i}$ based on their objective function evaluations. If the trial solution is better or equal than the current solution, it replaces the current solution, otherwise the current solution survives while the trial one is eliminated, as described below:

$$\mathbf{x}_{t+1,i} = \begin{cases} \mathbf{u}_{t,i} & \text{if } f(\mathbf{u}_{t,i}) \leq f(\mathbf{x}_{t,i}) \\ \mathbf{x}_{t,i} & \text{otherwise} \end{cases} \tag{5}$$

These variants of DE can be designated by the notation `DE/base/d/rec`, with `base` identifying the form of selection of the base solution, d representing the number of difference vectors used in differential mutation, and `rec` identifying the kind of recombination operator used to produce the offspring.

For instance, the basic DE, or canonical DE, known as `DE/rand/1/bin`, employs a random base solution, one difference vector, and binomial recombination. Other variants of this basic scheme are presented and discussed in [29, 10, 18, 6]. Table 1 summarizes some variants of DE with their respective notation.

Table 1 Some instances of the Differential Evolution algorithm.

Notation	Differential mutation equation
`DE/rand/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_0} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right)$
`DE/best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,best} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right)$
`DE/mean/1/bin`	$\mathbf{v}_{t,i} = \frac{1}{P}\sum_{l=1}^{P} \mathbf{x}_{t,l} + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right)$
`DE/rand-to-best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_0} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,r_0}\right) + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right)$
`DE/current-to-best/1/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,i} + \lambda\left(\mathbf{x}_{t,best} - \mathbf{x}_{t,i}\right) + F\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_2}\right)$
`DE/rand/2/bin`	$\mathbf{v}_{t,i} = \mathbf{x}_{t,r_0} + F_1\left(\mathbf{x}_{t,r_1} - \mathbf{x}_{t,r_3}\right) + F_2\left(\mathbf{x}_{t,r_2} - \mathbf{x}_{t,r_4}\right)$

3 A General Approach for Combinatorial Problems

In order to devise a DE-based algorithm for combinatorial optimization, the difference between two distinct solutions in the search space should be defined in a more meaningful way. The key idea in this chapter is to define the difference between two candidate solutions as a list of movements in the search space, as discussed next.

3.1 The Differential List of Movements

Local search heuristics in combinatorial optimization start with a complete solution, obtained at random or using any constructive heuristic, and iteratively refine this solution moving to another one in a neighborhood structure adequately defined for the problem. The basis of any local search method is thus the concept of a neighborhood structure:

Definition 1 (Neighborhood structure). *Let $\mathcal{S}$ be the search space of a given problem and* s *a solution in $\mathcal{S}$. A neighborhood structure is a set $\mathcal{N}(\mathrm{s}) \subset \mathcal{S}$ which associates to every solution $\mathrm{s} \in \mathcal{S}$ a solution $\mathrm{s}' \in \mathcal{N}(\mathrm{s})$. Therefore, $\mathcal{N}(\mathrm{s})$ denotes the set of solutions s' that can be obtained from* s *by using an elementary movement.*

The transition from s to s' is called a movement. These elementary movements depend on the problem at hand and the representation of candidate solutions. For permutation problems, this movement can be an insertion or swap operation, while in graph structures, a basic movement can be the addition or remotion of edges.

From the definition of neighborhood structure, comes the definition of a locally optimal solution:

Definition 2 (Local optimal solution). *A solution $\mathbf{s}^* \in \mathcal{S}$ is locally optimal (or local minimum) with respect to $\mathcal{N}(\mathbf{s}) \subset \mathcal{S}$ iff $\forall \mathbf{s}' \in \mathcal{N}(\mathbf{s}^*)$ we have $f(\mathbf{s}^*) \leq f(\mathbf{s}')$.*

We define the differences between pairs of individuals in differential mutation as a set of elementary movements in the search space of the problem. Therefore, the differential list of movements is built by finding the list of sequential movements that lead one solution to another. The definition of the differential list of movements (DLM), required for the proposed DE-based algorithm, is given below:

Definition 3 (List of movements). *A differential list of movements $M_{j\to i}$ is a list containing a sequence of valid movements m_k such that the application of these movements to a solution $\mathbf{s}_j \in \mathcal{S}$ leads to the solution $\mathbf{s}_i \in \mathcal{S}$.*

In this way, the “difference” between two candidate solutions is defined as being this list of movements:

$$M_{j\to i} \doteq \mathbf{s}_i \ominus \mathbf{s}_j \tag{6}$$

where $\ominus$ is a special binary minus operator that returns a list of movements $M_{j\to i}$ that represents a path from $\mathbf{s}_j$ towards $\mathbf{s}_i$. This list, in some sense, captures the differences between these two solutions.

The application of a list of movements to a given solution is defined as follows:

$$\mathbf{s}'_i = \mathbf{s}_i \oplus M_{a\to b} \tag{7}$$

where the binary operator $\oplus$ receives a valid solution and a list of movements, returning another solution.

Notice that with these definitions, the following relation is valid:

$$\mathbf{s}_i = \mathbf{s}_j \oplus M_{j\to i} = \mathbf{s}_j \oplus (\mathbf{s}_i \ominus \mathbf{s}_j) \tag{8}$$

The multiplication of the differential list of movements by a constant also needs to be defined. We present the following definition:

Definition 4. *The multiplication of the list of movements $M_{i\to j}$ by a constant $F \in [0,1]$, denoted as $F \otimes M_{i\to j}$, returns a list $M'_{i\to j}$ with $\lceil F \times |M_{i\to j}| \rceil$ randomly chosen movements of $M_{i\to j}$, at a random sequence, where $|M_{i\to j}|$ is the size of the list.*

Thus, the multiplication of the list of movements by a constant can be denoted, using the special binary multiplication operator $\otimes$, by:

$$M'_{i\to j} = F \otimes M_{i\to j} \tag{9}$$

Using these definitions, one can write the mutant vector as:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus F \otimes (\mathbf{x}_{t,r_1} \ominus \mathbf{x}_{t,r_2}) \tag{10}$$
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus F \otimes M_{r_2 \to r_1} \tag{11}$$
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus M'_{r_2 \to r_1} \tag{12}$$

These operations represent the generalization of the differential mutation equation (2) from continuous domains to combinatorial optimization problems.

3.2 Example

Suppose the permutation solutions:

$$\mathbf{x}_{t,r_1} = \begin{bmatrix} 1\ 4\ 5\ 2\ 3\ 7\ 9\ 10\ 6\ 8 \end{bmatrix}$$
$$\mathbf{x}_{t,r_2} = \begin{bmatrix} 1\ 5\ 3\ 4\ 2\ 7\ 6\ 8\ 10\ 9 \end{bmatrix}$$

The list $M_{r_2 \to r_1}$ is built by iteratively finding elementary movements that make $\mathbf{x}_{t,r_2}$ closer to $\mathbf{x}_{r_1}$ for a given neighborhood structure. For instance, considering a neighborhood structure defined by swap movements, we get:

$$M_{r_2 \to r_1} = (2,\ 4);\ (3,\ 4);\ (4,\ 5);\ (7,\ 10);\ (8,\ 9);\ (9,\ 10) \tag{13}$$

where $(i,\ j)$ means swapping the elements at positions i and j.

The reader can verify that by applying the movements in this list to the solution $\mathbf{x}_{t,r_2}$ we get $\mathbf{x}_{t,r_1}$. This list can be scaled by F and applied to the base solution

$$\mathbf{x}_{t,base} = \begin{bmatrix} 1\ 3\ 4\ 2\ 6\ 5\ 7\ 9\ 10\ 8 \end{bmatrix}$$

Assuming $F = 0.6$, we get

$$M'_{r_2 \to r_1} = (7,\ 10);\ (4,\ 5);\ (2,\ 4);\ (9,\ 10) \tag{14}$$

In this way

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus F \otimes (\mathbf{x}_{t,r_1} \ominus \mathbf{x}_{t,r_2}) \tag{15}$$
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus F \otimes M_{r_2 \to r_1} \tag{16}$$
$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus M'_{r_2 \to r_1} \tag{17}$$
$$\mathbf{v}_{t,i} = \begin{bmatrix} 1\ 2\ 3\ 6\ 4\ 5\ 8\ 9\ 10\ 7 \end{bmatrix} \tag{18}$$

3.3 Pseudocode

We conclude this section presenting the pseudocode of the proposed DE version for combinatorial problems.

```
1. Initialize the population $\{\mathbf{x}_{t,1}, \cdots, \mathbf{x}_{t,P}\}$;
2. For each $i, \ldots, P$ do:
```

a. Select $\mathbf{x}_{t,base}$;
b. Build the differential list of movements:

$$M_{r_2 \to r_1} \doteq \mathbf{x}_{t,r_1} \ominus \mathbf{x}_{t,r_2}$$

c. Generate the mutant vector using:

$$\mathbf{v}_{t,i} = \mathbf{x}_{t,base} \oplus F \otimes (\mathbf{x}_{t,r_1} \ominus \mathbf{x}_{t,r_2})$$

d. Apply local search operator to $\mathbf{v}_{t,i}$ (*optional*);
e. Generate the trial solution $\mathbf{u}_{t,i}$ performing recombination of $\mathbf{x}_{t,i}$ and $\mathbf{v}_{t,i}$ with probability CR, otherwise, the trial solution is equal to the mutant solution;
f. Perform competition between the trial solution $\mathbf{u}_{t,i}$ and the current solution $\mathbf{x}_{t,i}$:

$$\mathbf{x}_{t+1,i} = \begin{cases} \mathbf{u}_{t,i} & \text{if } f(\mathbf{u}_{t,i}) \leq f(\mathbf{x}_{t,i}) \\ \mathbf{x}_{t,i} & \text{otherwise} \end{cases}$$

3. If stopping criteria are met, stop and return the best solution, otherwise increment t and go to step 2.

4 Previous Work

The DE algorithm is originally applicable to continuous optimization problems because its search mechanism is based on perturbations built with difference vectors. However, when dealing with combinatorial optimization problems with symbolic variables, these arithmetic operations are neither applicable nor meaningful. In this section, permutation-based approaches for DE in the literature are reviewed.

4.1 The Permutation Matrix Approach

The permutation matrix (PM) approach was proposed by Price and Storn for the differential mutation operator in the discrete domain [29]. A permutation matrix $\mathbf{P}$ is a matrix that maps a permutation vector into another permutation vector. The permutation matrix that maps an integer permutation vector $\mathbf{x}_{t,r_2}$ into another integer permutation vector $\mathbf{x}_{t,r_1}$ is given by the relation:

$$\mathbf{x}_{t,r_1} = \mathbf{P}\mathbf{x}_{t,r_2} \tag{19}$$

One can say that a permutation matrix $\mathbf{P}$ is a matrix that "leads" $\mathbf{x}_{t,r_2}$ to $\mathbf{x}_{t,r_1}$, and, in the differential evolution method, this matrix is regarded as the "difference" obtained from these two candidate solutions. The analogous equation for the differential mutation is given by:

$$\mathbf{v}_{t,i} = \mathbf{P}_F \cdot \mathbf{x}_{t,base} \tag{20}$$

where $\mathbf{P}_F$ is a modified permutation matrix, scaled by the parameter F, here with the meaning of a probability, in order to perform a fraction of the permutation represented by the original permutation matrix $\mathbf{P}$ [29]. Therefore, the PM method applies some swaps to the base vector $\mathbf{x}_{t,base}$, randomly selected from the set of permutations represented by $\mathbf{P}$.

4.2 Adjacency Matrix Approach

The adjacency matrix (AM) approach is also presented in [29]. In this approach integer permutations are encoded as an adjacency matrix. The "difference" between two candidates solutions is a matrix Δ_{r_1,r_2} given by the exclusive-OR (XOR) logical operation of the adjacency matrices associated with $\mathbf{x}_{t,r_1}$ to $\mathbf{x}_{t,r_2}$:

$$\Delta_{r_1,r_2} = A_{r_1} \text{ XOR } A_{r_2} \tag{21}$$

This difference matrix is added to the adjacency matrix of the base solution. However, it is unlikely that valid adjacency matrices are obtained. In fact, many invalid solutions are produced, for this reason this approach requires suitable repair operators.

4.3 Relative Position Indexing Approach

The relative position indexing (RPI) approach is adopted for the differential evolution in some works, see [30, 24], and is applicable for permutation-based problems only. This approach transforms the elements of the integer permutation vector into the floating-point interval $[0,1]$, applying the differential mutation directly using the transformed values in the continuous domain. The resulting values are then converted back into integer domain using relative position indexing, as described in [15].

For instance, given three vectors, each one a permutation solution, randomly chosen from the current population:

$$\mathbf{x}_{t,r_1} = \begin{bmatrix} 4\ 2\ 1\ 5\ 3 \end{bmatrix}; \ \mathbf{x}_{t,r_2} = \begin{bmatrix} 1\ 3\ 4\ 5\ 2 \end{bmatrix}; \ \mathbf{x}_{t,r_3} = \begin{bmatrix} 1\ 2\ 5\ 3\ 4 \end{bmatrix}$$

The transformation into floating-point values is achieved by dividing each element of the vector by the largest one of them, in this case 5:

$$\hat{\mathbf{x}}_{t,r_1} = \begin{bmatrix} 0.8\ 0.4\ 0.2\ 1.0\ 0.6 \end{bmatrix}$$

$$\hat{\mathbf{x}}_{t,r_2} = \begin{bmatrix} 0.2\ 0.6\ 0.8\ 1.0\ 0.4 \end{bmatrix}$$

$$\hat{\mathbf{x}}_{t,r_3} = \begin{bmatrix} 0.2\ 0.4\ 1.0\ 0.6\ 0.8 \end{bmatrix}$$

and applying equation (2) with $F = 0.6$, $d = 1$, the mutant vector is given by:

$$\hat{\mathbf{v}}_{t,i} = \begin{bmatrix} 0.80\ 0.28\ 0.32\ 0.76\ 0.84 \end{bmatrix}$$

In order to convert the mutant vector back into integer domain, using RPI, replace the smallest floating-point value by the smallest integer value, and then replace the next smallest floating-point value by the next integer value, and so on until all elements have been converted. Doing this procedure for the mutant vector above, we obtain:

$$\mathbf{v}_{t,i} = \begin{bmatrix} 4 \ 1 \ 2 \ 3 \ 5 \end{bmatrix}$$

This approach always yields a feasible solution, except when two or more floating-point values are the same. When such an event occurs, the trial vector must be repaired or discarded.

4.4 Largest Order Value Approach

The largest order value (LOV) approach is based on random key representation. The method also operates on real-valued vectors, using LOV to convert an individual into a permutation vector $\boldsymbol{\pi}$. The conversion procedure begins by ranking all elements in $\mathbf{x}_{t,i}$ by descending order, obtaining the rank vector

$$\boldsymbol{\phi}_i = \begin{bmatrix} \phi_{i,1} \ \phi_{i,2} \ \dots \ \phi_{i,N} \end{bmatrix}$$

Then assign $\pi_{t,i,\phi_{i,k}} = k$ for $k = 1, \dots, N$. For instance, consider the real-valued vector

$$\mathbf{x}_{t,i} = \begin{bmatrix} 1.36 \ 3.85 \ 2.55 \ 0.63 \ 0.51 \end{bmatrix}$$

The rank vector is given by:

$$\boldsymbol{\phi}_i = \begin{bmatrix} 3 \ 1 \ 2 \ 4 \ 5 \end{bmatrix}$$

since 3.85 is the largest value, $\phi_{i,2} = 1$, and 0.51 is the smallest value, thus $\phi_{i,5} = 5$. Varying k from 1 to N, we get:

$$\begin{aligned}
\pi_{t,i,\phi_{i,1}} &= \pi_{t,i,3} = 1; \\
\pi_{t,i,\phi_{i,2}} &= \pi_{t,i,1} = 2; \\
\pi_{t,i,\phi_{i,3}} &= \pi_{t,i,2} = 3; \\
\pi_{t,i,\phi_{i,4}} &= \pi_{t,i,4} = 4; \\
\pi_{t,i,\phi_{i,5}} &= \pi_{t,i,5} = 5;
\end{aligned}$$

Therefore

$$\boldsymbol{\pi}_{t,i} = \begin{bmatrix} 2 \ 3 \ 1 \ 4 \ 5 \end{bmatrix}$$

4.5 Forward Backward Transformation Approach

The forward backward transformation (FBT) approach, presented in [22, 20, 21], also transforms a given permutation vector into the floating-point domain, using the *forward transformation* given by:

$$\hat{\mathbf{x}}_{t,i} = -1 + \alpha \mathbf{x}_{t,i} \tag{22}$$

where α is a small number. In [21], the author suggests the ratio $\alpha = 500/999$.

The differential mutation equation given by (2) is applied using the transformed values, and the result is converted into integer values using the *backward transformation* given by:

$$\mathbf{x}_{t,i} = \text{round}\left[\frac{1}{\alpha}(1 + \hat{\mathbf{x}}_{t,i})\right] \tag{23}$$

Although the backward transformation effectively produces integer vectors, it usually generates some invalid solutions, i.e., integer vectors that do not represent valid permutations. The invalid solutions must be repaired using suitable repairing methods, see [21] for a description of some repairing methods that can be used.

4.6 Discussion

Evolutionary algorithms are recognized as very general and robust optimization methods, in the sense that they are in principle applicable to a wide range of optimization problems without requiring strong premises about the problem. Provided that a suitable representation for the candidate solutions of the problem and operators for this representation are designed, one can apply the designed evolutionary algorithm to solve the problem. This class of methods are viewed as general-purpose tools, but they require at least one premise for functioning properly and efficiently in a given problem, that principle known as *weak locality*:

Definition 5 (Weak locality). *The value of the objective function at a given point is correlated to the values of the objective function at points within its vicinity.*

The weak locality is even more important in DE, since small differences translate into small perturbations applied to the base vector, and therefore a more localized search. The methods reviewed in this section that are based on mapping floating point values to integer values can violate this principle, making the optimization problem harder for the differential evolution algorithm. As an example, consider the RPI method and the vector

$$\hat{\mathbf{x}}_{base} = \begin{bmatrix} 0.80 \ 0.28 \ 0.32 \ 0.76 \ 0.84 \end{bmatrix}$$

It is possible to define a small perturbation in the continuous domain that would cause a great change in the permutation domain, for instance:

$$\hat{\mathbf{x}}_{base} = \begin{bmatrix} 0.80 \\ 0.28 \\ 0.32 \\ 0.76 \\ 0.84 \end{bmatrix} + \begin{bmatrix} 0.03 \\ 0.02 \\ -0.03 \\ 0.05 \\ -0.04 \end{bmatrix} = \begin{bmatrix} 0.83 \\ 0.30 \\ 0.29 \\ 0.81 \\ 0.80 \end{bmatrix}$$

The base solution represents the permutation $\left[4\ 1\ 2\ 3\ 5\right]$, whereas after the application of a small perturbation we get the permutation $\left[5\ 2\ 1\ 4\ 3\right]$, which is a very different permutation. Thus close solutions in the continuous domain might be distant in the permutation search space, which is detrimental to the self-adaptive nature of the differential mutation. It is possible to design similar examples for all the mapping approaches, namely, RPI, LOV and FBT methods.

Additionally, mapping approaches in practice reduce the differential mutation operator to a shuffling operator, not taking advantage of the good characteristics that this operator presents in continuous domains.

Finally, the adjacency and permutation matrices have other drawbacks. The AM approach can generate many invalid solutions, requiring suitable and carefully designed repairing methods. The PM method can be viewed as a particular case of the proposed approach using a movement list. In fact, the movement list using swap movements is equivalent to the permutation matrix, only using an alternate data structure. However, PM method is applicable only to permutation problems, while the movement list can be generalized to other contexts.

In summary, we remark the following advantages of the proposed approach:

- it does not require any repairing operators;
- it is general and applicable to other combinatorial optimization problems, for instance the PM approach can be seen as a special case of the proposed approach;
- it is flexible, since it can employ different movements for specific problems;
- it preserves the search mechanism and principles of DE for discrete domains;
- it forms a general and unifying framework for DE in combinatorial optimization;

5 Flow Shop Scheduling

5.1 Problem Statement

In the permutation FSSP (PFSSP), solutions are represented by the permutation of n jobs, i.e., $\boldsymbol{\pi}_{t,i} = \left[\pi_{t,i,1},\ \pi_{t,i,2},\ \ldots,\ \pi_{t,i,n}\right]$ and each job will be sequenced through m machines. Jobs, once initiated, can not be interrupted (without preemption). Thus, given the processing time p_{jk} for the job $j \in \{1, 2, ..., n\}$ on the machine $k \in \{1, 2, ..., m\}$, the PFSSP can be stated as follows:

Definition 6 (Statement of the PFSSP). *Find the best permutation of n jobs $\boldsymbol{\pi}^* = \left[\pi_1^*,\ \pi_2^*,\ \ldots,\ \pi_n^*\right]$ to be processed on m machines that minimizes the makespan.*

Let $C(\pi_j, m)$ denote the completion time of the job π_j on the machine m. Given the job permutation $\boldsymbol{\pi}$, the calculation of completion time for the n-job m-machine problem is given as follows.

$$C(\pi_1, 1) = p_{\pi_1,1} \tag{24}$$
$$C(\pi_j, 1) = C(\pi_{j-1}, 1) + p_{\pi_j,1}, \; j = 2, \ldots, n \tag{25}$$
$$C(\pi_1, k) = C(\pi_1, k-1) + p_{\pi_1,k}, \; k = 2, \ldots, m \tag{26}$$
$$C(\pi_j, k) = \max\left\{C(\pi_{j-1}, k), C(\pi_j, k-1) + p_{\pi_j,k}\right\}, \; j = 2, \ldots, n, \; k = 2, \ldots, m \tag{27}$$

Therefore, the makespan of a permutation $\boldsymbol{\pi}$ can be formally defined as the completion time of the last job π_n on the last machine m, i.e., $C_{\max}(\boldsymbol{\pi}) = C(\pi_n, m)$. The PFSSP with the makespan criterion corresponds to finding a permutation $\boldsymbol{\pi}^*$ in the set of all permutations Π such that:

$$C_{\max}(\boldsymbol{\pi}^*) \leq C(\pi_n, m), \;\; \forall \boldsymbol{\pi} \in \Pi \tag{28}$$

5.2 Experimental Setup

The dDE algorithm was coded in Visual C++ and executed on an Intel Celeron 2.13GHz PC. It was applied to the 120 instances of Taillard [37] ranging from 20 jobs with 5 machines to 500 jobs with 20 machines. Regarding the parameters, the population size is fixed in 10 individuals, consistent with [25] and a pool with three fixed parameter settings is used. Each time an individual goes through the variation phase (*mutation* and *crossover*), one of the combinations is selected randomly. The three combinations are:

1. $[F = 1, CR = 0.1]$;
2. $[F = 1, CR = 0.9]$;
3. $[F = 0.8, CR = 0.2]$.

This strategy for choosing control parameters was proposed by [38] and has shown good performance in a number of test functions. The initial population is generated randomly. The crossover operator used was the one proposed in [8], known as ordered crossover (OX). In order to give a wider variety of movements to the algorithm, two different mutation strategies were employed in the experiments. The `DE/rand/1` strategy is performed with probability 0.8 and `DE/best/1` with probability 0.2. With this setting no effort has been devoted to adjusting dDE parameters.

Here it was also employed the referenced local search (RLS), see [25], applied to the best individual at each generation. In the case of the best individual being the same one in the previous generation, a random individual is chosen to go through local search.

To perform a comparisson of dDE with some other best performing algorithms from the literature, we adopt the results reported in [34] and [25]. In [34] the authors have proposed the algorithms IG_RS and IG_RS$_{LS}$ and have implemented NEHT [36], GA_RMA [33], SA_OP [23], SPIRIT [39], HGA_RMA [33], GA_CHEN [7], GA_REEV [32], GA_MIT [19], ILS [35], GA_AA [2], M-MMAS and PACO from [31]. In [25] the authors have proposed and implemented the DDE$_{RLS}$ algorithm.

The results are reported in terms of the average relative percentage deviation, which is computed as follows:

$$\varDelta_{avg} = \frac{1}{R}\sum_{i=1}^{R}\left(\frac{(H_i - H_{ref}) \times 100}{H_{ref}}\right) \tag{29}$$

where R is the number of runs conducted for each problem instance ($R = 5$, consistent with [34] and [25]), H_i is the makespan generated by one of the R replications of the metaheuristic algorithm and H_{ref} is the bound value provided by [37].

The stop criteria was set to $n \times (m/2) \times t$, where t is a time in ms. [34] have used a PC with a Athlon XP 1600+ processor (1.4GHz) and $t = 60ms$. [25] have used a Intel Pentium IV 3.0GHz and $t = 30ms$. Considering the differences between the processors, taking $t = 40ms$ for dDE would be reasonable for comparison with the results of other authors. Table 5.2 summarizes the results.

Table 2 Average relative percentage deviation over the optimal solution value or the best known upper bound for Taillard instances when the algorithms are evaluated with the termination criterion set as $n \times (m/2) \times 60ms$, except for dDE where it was set as $n \times (m/2) \times 40ms$ and DDE_{RLS} where it was set as $n \times (m/2) \times 30ms$.

n × m	NEHT	GA_RMA	HGA_RMA	SA_OP	SPIRIT	GA_CHEN	GA_REEV	GA_MIT
20 × 5	3.35	0.26	0.04	1.09	4.33	4.15	0.62	0.8
20 × 10	5.02	0.73	0.13	2.63	6.07	5.18	2.04	2.14
20 × 20	3.73	0.43	0.09	2.38	4.44	4.26	1.32	1.75
50 × 5	0.84	0.07	0.02	0.52	2.19	2.03	0.21	0.3
50 × 10	5.12	1.71	0.72	3.51	6.04	6.54	2.06	3.55
50 × 20	6.26	2.74	1.28	4.52	7.63	7.74	3.56	5.09
100 × 5	0.46	0.07	0.02	0.3	1.06	1.35	0.17	0.27
100 ×10	2.13	0.62	0.29	1.48	3.01	3.8	0.85	1.63
100 ×20	5.23	2.75	1.66	4.63	6.74	8.15	3.41	4.87
200 ×10	1.43	0.43	0.2	1.01	2.07	2.76	0.55	1.14
200 ×20	4.41	2.31	1.48	3.81	4.97	7.24	2.84	4.18
500 ×20	2.24	1.4	0.96	2.52	12.58	4.72	1.66	3.34
Average	3.19	1.25	0.67	2.47	5.07	4.86	1.66	2.61

n × m	ILS	GA_AA	M-MMAS	PACO	IG_RS	IG_RS_LS	DDE_RLS	**dDE**
20 × 5	0.49	0.94	0.04	0.21	0.04	0.04	0.04	**0.29**
20 × 10	0.59	1.54	0.15	0.37	0.25	0.06	0.01	**1.45**
20 × 20	0.36	1.43	0.06	0.24	0.21	0.03	0.02	**1.03**
50 × 5	0.2	0.36	0.03	0.01	0.04	0	0	**0.13**
50 × 10	1.48	3.72	1.4	0.85	1.06	0.56	0.45	**1.89**
50 × 20	2.2	4.69	2.18	1.59	1.82	0.94	0.66	**3.03**
100 × 5	0.18	0.32	0.04	0.03	0.05	0.01	0	**0.12**
100 ×10	0.68	1.72	0.47	0.27	0.39	0.2	0.15	**0.74**
100 ×20	2.55	4.91	2.59	2.09	2.04	1.3	0.98	**3.70**
200 ×10	0.56	1.27	0.23	0.27	0.34	0.12	0.07	**0.61**
200 ×20	2.24	4.21	2.26	1.92	1.99	1.26	0.99	**3.58**
500 ×20	1.25	2.23	1.15	1.09	1.13	0.78	0.49	**2.88**
Average	1.17	2.28	1.04	0.84	0.91	0.52	0.38	**1.77**

5.3 Discussion

Comparing the averages obtained by the sixteen algorithms the dDE occupies the 10th position. The relative percentage deviation generated by dDE (1.77%) was lower then those generated by GA_AA (2.28%), SA_OP (2.47%), GA_MIT (2.61%), NEHT (3.19%), GA_CHEN (4.86%) and SPIRIT (5.07%). Despite of this result obtained by dDE, it can be considered competitive when compared to GA_RMA (1.25%), GA_REEV (1.66%) and ILS (1.17%), which have already been shown to be fairly good algorithms in the literature [25].

The best performing algorithms in this experiment were M-MMAS (1.04%), PACO (0.84%), IG_RS (0.91%), IG_RS$_{LS}$ (0.52%) and DDE$_{RLS}$ (0.38%). M-MMAS and PACO are two ant colony algorithms, IG_RS and IG_RS$_{LS}$ are iterated greedy algorithms and DDE$_{RLS}$ is another discrete version of Differential Evolution.

Regardless of the 10th place, the dDE algorithm presented here can still be considered a promising and competitive tool for PFSSP. It is important to remark that dDE is a general approach and no effort was spent on setting its parameters. Regarding the five best performing algorithms, they all rely in some way on the NEH initialization heuristic, which was not adopted in dDE and could have benefited the dDE as well. The differential evolution used here is relatively simple, in contrast to other methods. There is a number of sophisticated techniques such as self-adaptation, re-initialization and path-relinking that could be easily implemented in the dDE framework in order to improve its performance.

6 Conclusion

This chapter has presented a general framework of Differential Evolution for combinatorial optimization problems. The proposed approach is based on defining the differences between pairs of individuals as a list of elementary movements in the discrete search space. This approach is general and can be used in principle for other combinatorial problems and for different kinds of neighborhood structures. The method was applied to the 120 Taillard instances of the PFSSP, and compared against the results obtained by other metaheuristic algorithms in the literature. Although relying only on the differential mutation and the local search performed on the best individual, dDE ranks fairly well against more sophisticated metaheuristics. The results are promising and illustrate the applicability of the proposed approach for combinatorial optimization using differential evolution.

References

1. Abbass, H.A., Sarkar, R., Newton, C.: A pareto differential evolution approach to vector optimisation problems. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, pp. 971–978. IEEE Press (2001)

2. Aldowaisan, T., Allahverdi, A.: New heuristics for no-wait flowshops to minimize makespan. Computers & Operational Research 30, 1219–1231 (2003)
3. Allahverdi, A., Ng, C.T., Cheng, T.C.E., Kovalyov, M.Y.: A survey of scheduling problems with setup times or costs. European Journal of Operational Research 187(3), 985–1032 (2008)
4. Baker, K.R.: Introduction to Sequencing and Scheduling. Wiley, New York (1974)
5. Batista, L.S., Guimarães, F.G., Ramírez, J.A.: A differential mutation operator for the archive population of multi-objective evolutionary algorithms. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, pp. 1108–1115. IEEE Press (2009)
6. Chakraborty, U.K. (ed.): Advances in Differential Evolution. SCI, vol. 143. Springer, Heidelberg (2008)
7. Chen, C.-L., Vempati, V.S., Aljaber, N.: An application of genetic algorithms for flow shop problems. European Journal of Operational Research 80(2), 389–396 (1995)
8. Davis, L.: Job shop scheduling with genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 136–140. L. Erlbaum Associates Inc., Hillsdale (1985)
9. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer (2003)
10. Feoktistov, V.: Differential Evolution: In Search of Solutions. Springer Optimization and Its Applications, 1st edn. Springer (October 2006)
11. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman (1979)
12. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 117–129 (1976)
13. Gong, W., Cai, Z.: A multiobjective differential evolution algorithm for constrained optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, pp. 181–188. IEEE Press (June 2008)
14. Kim, H.-K., Chong, J.-K., Park, K.-Y., Lowther, D.A.: Differential evolution strategy for constrained global optimization and application to practical engineering problems. IEEE Transactions on Magnetics 43(4), 1565–1568 (2007)
15. Lichtblau, D.: Relative position indexing approach. In: Onwubolu, G.C., Davendra, D. (eds.) Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. SCI, vol. 175, ch. 4, pp. 81–120. Springer (2009)
16. Linn, R., Zhang, W.: Hybrid flow shop scheduling: a survey. Computers & Industrial Engineering 37(1-2), 57–61 (1999)
17. Madavan, N.K.: Multiobjective optimization using a Pareto differential evolution approach. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, vol. 2, pp. 1145–1150. IEEE Press (May 2002)
18. Mezura-Montes, E., Velazquez-Reyes, J., Coello Coello, C.A.: A comparative study of differential evolution variants for global optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO, pp. 485–492. ACM (2006)
19. Murata, T., Ishibuchi, H., Tanaka, H.: Genetic algorithms for flowshop scheduling problems. Computers & Industrial Engineering 30, 1061–1071 (1996)
20. Onwubolu, G.C.: Design of hybrid differential evolution and group method of data handling networks for modeling and prediction. Information Sciences 178, 3616–3634 (2008)
21. Onwubolu, G.C., Davendra, D. (eds.): Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. SCI, vol. 175. Springer (2009)

22. Onwubolu, G.C., Davendra, D.: Scheduling flow shops using differential evolution algorithm. European Journal of Operational Research 171(2), 674–692 (2006)
23. Osman, I.H., Potts, C.N.: Simulated annealing for permutation flow-shop scheduling. Omega 17(6), 551–557 (1989)
24. Pan, Q.K., Wang, L., Qian, B.: A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problem. Computers & Operations Research 36(8), 2498–2511 (2009)
25. Pan, Q.-K., Tasgetiren, M.F., Liang, Y.-C.: A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers & Industrial Engineering 55, 795–816 (2008)
26. Pinedo, M.: Scheduling: Theory, Algorithms and Systems. Prentice-Hall (2002)
27. Price, K.V.: An introduction to differential evolution. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimisation. Advanced Topics in Computer Science, pp. 79–108. McGraw-Hill (1999)
28. Price, K.V., Storn, R.M.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11(4), 341–359 (1997)
29. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization, 1st edn. Natural Computing Series. Springer (December 2005)
30. Qian, B., Wang, L., Hu, R., Wang, W.-L., Huang, D.-X., Wang, X.: A hybrid differential evolution method for permutation flow-shop scheduling. The International Journal of Advanced Manufacturing Technology 38, 757–777 (2008)
31. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155(2), 426–438 (2004)
32. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Computers & Operational Research 22, 5–13 (1995)
33. Ruiz, R., Maroto, C., Alcaraz, J.: Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34(5), 461–476 (2006)
34. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177, 2033–2049 (2007)
35. Stützle, T.: Applying iterated local search to the permutation flow shop problem. Technical report, AIDA-98-04, FG Intellektik, FB Informatik, TU Darmstadt (1998)
36. Taillard, E.: Some efficient heuristic methods for the flow shop sequencing problem. European Journal of Operational Research 47(1), 65–74 (1990)
37. Taillard, E.: Benchmarks for basic scheduling problems. European Journal Of Operational Research 64(2), 278–285 (1993)
38. Wang, Y., Cai, Z., Zhang, Q.: Differential evolution with composite trial vector generation strategies and control parameters. IEEE Transactions on Evolutionary Computation 15(1), 55–66 (2011)
39. Widmer, M., Hertz, A.: A new heuristic method for the flow shop sequencing problem. European Journal of Operational Research 41(2), 186–193 (1989)
40. Xue, F., Sanderson, A.C., Graves, R.J.: Pareto-based multi-objective differential evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC, vol. 2, pp. 862–869. IEEE Press (2003)