Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem with Total Flowtime Criterion

Valentino Santucci, Marco Baioletti, and Alfredo Milani, *Member, IEEE*

***Abstract*—This work introduces an original algebraic approach to Differential Evolution algorithms for combinatorial search spaces. An abstract algebraic differential mutation for generic combinatorial spaces is defined by exploiting the concept of finitely generated group. This operator is specialized for the permutations space by means of an original randomized bubble sort algorithm. Then, a discrete Differential Evolution algorithm is derived for permutation problems and it is applied to the permutation flowshop scheduling problem with the total flowtime criterion. Other relevant components of the proposed algorithm are: a crossover operator for permutations, a novel biased selection strategy, a heuristic-based initialization and a memetic restart procedure. Extensive experimental tests have been performed on a widely accepted benchmark suite in order to analyze the dynamics of the proposed approach and to compare it with the state-of-the-art algorithms. The experimental results clearly show that the proposed algorithm reaches state-of-the-art performances and, most remarkably, it is able to find some new best known results. Furthermore, the experimental analysis on the impact of the algorithmic components shows that the two main contributions of this work, i.e., the discrete differential mutation and the biased selection operator, greatly contribute to the overall performance of the algorithm.**

***Index Terms*—Differential Evolution, Permutations Space, Permutation Flowshop Scheduling, Algebraic Differential Mutation.**

I. Introduction

The Permutation Flowshop Scheduling Problem (PFSP) is one of the most studied scheduling problems in computer science and operational research both for its practical applications and its theoretical aspects [1].

In PFSP, a set $J = \{1, \dots, n\}$ of n jobs has to be scheduled on a set $M = \{1, \dots, m\}$ of m machines. M is provided with a fixed order and each job has to visit, in order, all the machines. The given processing time of job $j \in J$ on machine $i \in M$ is $p_{i,j}$. Every machine can process only one job at a time. Preemption and job-passing are not allowed. PFSP requires to find an ordering on the jobs in order to optimize a given objective function expressed in terms of the processing times. Formally, a PFSP solution π is the permutation of jobs $\langle \pi(1), \dots, \pi(n) \rangle$, where $\pi(k) \in J$, with $1 \leq k \leq n$, indicates the job at position k in π. In this work, we consider the Total Flow Time (TFT) criterion, i.e., the minimization of the objective function

$$f(\pi) = \sum_{j=1}^{n} c(m, \pi(j)) \tag{1}$$

where $c(i, \pi(k))$ is the completion time of the job $\pi(k)$ on the machine i and it is recursively defined as

$$c(i, \pi(k)) = p_{i,\pi(k)} + \max\{c(i, \pi(k-1)), c(i-1, \pi(k))\}$$

when $i > 1$ and $k > 1$, while the terminal cases are:

$$\begin{array}{ll} c(i, \pi(k)) = p_{i,\pi(k)} & \text{if } i = k = 1, \\ c(i, \pi(k)) = p_{i,\pi(k)} + c(i, \pi(k-1)) & \text{if } i = 1 \text{ and } k > 1, \\ c(i, \pi(k)) = p_{i,\pi(k)} + c(i-1, \pi(k)) & \text{if } i > 1 \text{ and } k = 1. \end{array}$$

Compared with the classical makespan criterion, TFT can be seen as a customers oriented objective and it has recently received an increasing interest [2], [3]. In the following we will refer to PFSP with the TFT criterion as PFSP-TFT.

PFSP-TFT has been proved to be NP-hard for $m \geq 2$ [1]. Since then, many researches have been devoted to finding high quality solutions by means of heuristic or meta-heuristic approaches [2], [3]. Among them, also Differential Evolution (DE) schemes have been applied to PFSP problems. However, the original DE [4] scheme addresses continuous optimization problems where solutions are represented by numerical vectors. Therefore, all the DE applications to PFSP proposed in the literature adopt some transformation scheme to encode a permutation as a numerical vector (see for example [5], [6]). A drawback of this approach is that a single permutation of jobs can be represented by a potentially infinite number of continuous solutions, thus inducing a one-to-many mapping from the phenotypic space (jobs permutations) to the genotypic space (numerical vectors). As a consequence, large plateaus are introduced in the search landscape and this is probably the main reason of the poor performances of these DE schemes on PFSP problems.

In this paper, we propose a new fully discrete DE for Permutation spaces (DEP) that directly represents solutions as permutations. Therefore, no distinction between phenotypic and genotypic spaces is made. An initial work in this direction has been already presented in [7].

DEP is conceived around an algebraic design of the differential mutation operator, which is generally considered the key component of DE [8]. Classical differential mutation works in numerical spaces by exploiting the self-adaptive distribution

The authors are all with the Department of Mathematics and Computer Science, University of Perugia, via Vanvitelli 1, 06123 Perugia, Italy. Alfredo Milani is also with the Department of Computer Science, Hong Kong Baptist University, Kowloon Tong, Hong Kong SAR China. Email addresses: valentino.santucci@dmi.unipg.it, marco.baioletti@unipg.it, alfredo.milani@unipg.it.

of the numerical differences among the solutions in the DE population [8]. However, the concept of solutions difference cannot be straightforwardly transferred to a combinatorial search space. To address this issue, we introduce an abstract and algebraic-based scheme that allows to define a difference operator between combinatorial objects and, consequently, to design a discrete differential mutation operator which inherits most of the properties of its numerical counterpart. The only requirement we impose is that the combinatorial search space must be representable by means of a finitely generated group. This is the case of the permutations search space of PFSP problems.

The abstract algebraic differential mutation operator has been specialized for the symmetric group of permutations by means of an original randomized bubble sort (RandBS) algorithm. RandBS has been also studied from the point of view of the complexity and it constitutes an additional result of this work.

In order to finalize the design of DEP, also other algorithmic components have been adopted or introduced: a crossover operator working on permutations, a biased selection scheme, a heuristic-based initialization, a restart procedure, and a local search method.

DEP has been validated by means of a broad experimental analysis aiming to: (i) investigate the impact of the various algorithmic components on the search, and (ii) compare DEP with the state-of-the-art algorithms for PFSP-TFT. The experiments have been held on widely accepted benchmark suites and statistical analyses have been conducted on the obtained results. Then, discussions are provided in order to show when and how the proposed approach allows to reach remarkable, or at least competitive, results both regarding its peak performances and its robustness.

The rest of the paper is organized as follows. In Section II, a review of the literature on PFSP-TFT algorithms is provided. The main scheme of DEP, together with a brief overview of the DE proposals for permutation spaces found in literature, are described in Section III. Section IV introduces the abstract algebraic differential mutation operator, its specialization for the permutations space, and presents the randomized bubble sort algorithm. Section V describes the other algorithmic components of DEP. The experimental analysis is provided in Section VI. Finally, conclusions are drawn in Section VII where future lines of research are also depicted.

II. Related Work

As in other combinatorial problems, three main approaches can be considered to solve PFSP-TFT: exact algorithms, heuristic methods, and meta-heuristics.

Since PFSP-TFT is NP-hard, algorithms that guarantee to find the optimal schedule, for instance by using the branch and bound method, are practical only for the instances with no more than 15 jobs [3].

To overcome the limits of the exact algorithms, heuristic methods have been proposed. These techniques generally work by iteratively adding jobs to a partial schedule until it becomes a complete solution. For this reason they are usually referred to as "constructive heuristics" [2]. One of the most popular heuristic for PFSP-TFT is the Liu-Reeves procedure LR [9]. LR allows to regulate the trade-off between efficiency and effectiveness by means of a parameter that is usually set to n/m (in this case, we will refer to it as LR(n/m)). Heuristic methods allow to handle larger instances, but, unfortunately, they show poor performances for 50 or more jobs [3]. Therefore, they are usually employed only to build initial solutions for the more powerful meta-heuristic methods.

Among the meta-heuristics, three main classes of algorithms can be recognized: local search schemes, evolutionary algorithms, and hybrid methods.

Local search schemes iteratively improve a single incumbent solution by alternating the phases of neighborhood search and random perturbation. For PFSP problems, usually the neighborhood structure of the search space is built with insertion or interchange moves [10]. The neighborhood search phase starts from a seed solution and iteratively updates it with an improving neighbor until a local optimum is reached. Then, a random perturbation is performed to (hopefully) escape from the local optimum and the local search is iterated again. Recently, two effective local search methods for PFSP-TFT, namely Iterated Local Search (ILS) and Iterated Greedy Algorithm (IGA), have been proposed in [11]. ILS and IGA share many features: they both build the initial solution using LR(n/m), they adopt an acceptance criterion of "simulated annealing type" (not only improving solutions, but also slightly worsening ones may be accepted), and they employ a neighborhood search (namely, iRZ [12]) based on the insertion neighborhood. Their difference resides in the perturbation scheme. While ILS blindly performs a small number of random insertion moves, IGA adopts a greedy destruction-construction scheme [13] guided by partial fitness evaluations.

Another important local search scheme is Variable Neighborhood Search (VNS), which introduces the idea of neighborhood change by alternating local search phases using different neighborhoods. Hence, when a local optimum is reached for the current neighborhood, a new local search on another neighborhood allows to continue the global search. In a recent work [14], six different VNS algorithms have been successfully applied to PFSP-TFT. All the six VNSs are based on the insertion and interchange neighborhoods, employ a perturbation procedure to escape from the local optima common to both neighborhoods, and build the initial solution using LR(n/m). They differ in how the neighborhood searches are combined. The most performing variant is VNS_4 that iteratively alternates a full interchange neighborhood search and one iteration of the insertion neighborhood search.

Differently from the local search schemes, evolutionary algorithms use Darwinian, statistical or swarm-intelligence principles to evolve a population of solutions. Often, evolutionary algorithms are hybridized with local search schemes as in the two recent PFSP-TFT state-of-the-art meta-heuristics, AGA [15] and HGM-EDA [3]. Both adopt LR(n/m) and extensively use local search.

AGA is an asynchronous genetic algorithm which uses a modified VNS procedure parameterized by an integer number,

called "power", indicating the depth of the local search. Just before applying the crossover, the two selected parents are improved using VNS with two different power values in order to diversify them. Then, the two produced offsprings are improved again by using VNS, this time both with the maximum allowed power.

HGM-EDA is composed by two evolution stages. The first stage is performed by a new estimation of distribution algorithm, i.e., GM-EDA, while in the second stage a VNS scheme similar to VNS_4 is employed to improve the best solution obtained by GM-EDA. The novelty of this work resides in the probabilistic model used in the first stage. Indeed, GM-EDA adopts the generalized Mallows model for permutations and introduces estimation and sampling procedures for it. However, the experimental results reported in [3] show that GM-EDA is able to reach state-of-the-art results only in its hybridized variant, i.e., HGM-EDA.

III. Differential Evolution for Permutations

The classical Differential Evolution (DE) [4], [16] is a popular and effective evolutionary algorithm [17], [18] for continuous optimization that iteratively employs the following genetic operators: differential mutation, crossover and selection. Despite the impressive performances in numerical optimization, DE applications to combinatorial problems, and particularly to permutation-based problems, are still unsatisfactory. Indeed, the vast majority of DE algorithms for the PFSP problem adopts some transformation scheme to encode permutations as numerical vectors. Therefore, classical DE is used to directly evolve the solutions which are decoded to permutations only in the evaluation step. For instance, the "random key" technique [19] decodes a numerical vector by sorting its components and using the indices of the sorted keys to represent a permutation. This and other similar techniques have been used in the DE schemes reported in [6] and, more recently, in [5], [20], [21].

To the best of our knowledge, the only DE scheme in literature that directly handles permutations representations is the Geometric DE (GDE) proposed in [22]. However, GDE, in its permutation-based form, has been applied only to the Traveling Salesman Problem (TSP) obtaining poor performances.

The differential mutation is generally considered the key component of DE [8] and it has been argued that it confers to DE the "contours matching" property (term coined by Price et al. in [8]), i.e., it allows numerical DE to automatically adapt both mutation step size and direction to the search space. Conversely, our approach mainly relies on an algebraic design of the differential mutation operator that allows to directly handle permutations in a way that is consistent with its numerical counterpart.

The main scheme of our DE for Permutation spaces (DEP) is outlined in Fig. 1 and resembles that of classical DE.

DEP directly evolves a population $\{\pi_1, \ldots, \pi_N\}$ of N permutations. The three genetic operators of mutation, crossover and selection mainly differ from the classical ones for their internal implementations. For every population individual π_i,

```
1: Initialize Population {π_1, ..., π_N}
2: while num_fit_evals ≤ max_fit_evals do
3:     for i ← 1 to N do
4:         ν_i ← DifferentialMutation(i, {π_1, ..., π_N}, F)
5:         (υ'_i, υ''_i) ← Crossover(π_i, ν_i)
6:         Evaluate f(υ'_i) and f(υ''_i)
7:     end for
8:     for i ← 1 to N do
9:         π_i ← Selection(π_i, υ'_i, υ''_i, θ)
10:    end for
11:    if restart criterion is satisfied then
12:        Optionally perform a Local Search on π_best
13:        Restart Population
14:    end if
15: end while
```

Fig. 1. Differential Evolution for Permutations

the differential mutation operator builds a mutant ν_i by using three individuals randomly selected from the current population and the scale factor parameter $F \in (0, 1]$. Its working scheme is described and motivated in Section IV. The adopted crossover is parameterless and, given π_i and ν_i, produces two offsprings, i.e., υ'_i and υ''_i, which both undergo selection with π_i. The selection operator has been extended by introducing the real-valued parameter θ that regulates the selective pressure of DEP. The employed crossover and the proposed selection are both described in Section V. Finally, in order to improve the quality of the results, three other aspects are considered: a possibly guided initialization, a restart mechanism, and the use of a local search refinement. Also these DEP components are described in Section V.

IV. Differential Mutation for Permutations

In the most common variant of differential mutation (usually denoted as "rand/1"), for each population individual x_i, a mutant v_i is generated according to

$$v_i \leftarrow x_{r_0} \oplus F \odot (x_{r_1} \ominus x_{r_2}) \tag{2}$$

where the scalar F is the DE scale factor parameter that usually lies in $(0, 1]$, and x_{r_0}, x_{r_1}, x_{r_2} are three randomly chosen distinct population individuals, all different from x_i. Moreover, in the numerical DE case, the operators $\oplus$, $\ominus$, $\odot$ are the usual vectorial operations of $\mathbb{R}^n$.

In the following we introduce a differential mutation operator for the permutations space that is consistent with the classical numerical definition. First, in Section IV-A, after recalling some preliminary concepts of group theory, we propose and motivate an abstract differential mutation scheme for generic combinatorial spaces. Then, in Section IV-B, we provide an implementation of this scheme for the permutations space.

A. Differential Mutation in Finitely Generated Groups

The key point behind the classical numerical differential mutation is the dichotomous interpretation of a vector in the Euclidean space. Indeed, it can be interpreted either as a point in the space, thus a DE candidate solution, and as a free vector (without point of application) connecting two points/solutions. In this sense, equation (2) can be interpreted as follows. $x_{r_1} -$

x_{r_2} is the free vector $\vec{d}$ that, if applied to x_{r_2}, produces x_{r_1}, i.e., $x_{r_2} + \vec{d} = x_{r_1}$. Then, the magnitude of $\vec{d}$ is scaled by $F \in (0, 1]$, and, finally, the scaled vector $F \cdot \vec{d}$ is applied to x_{r_0} producing the mutant v_i.

Here we provide an abstract scheme that allows to apply the differential mutation to a generic discrete search space in a way that is consistent with the aforementioned interpretation. The only condition we require is that the solutions in the search space form a finitely generated group.

Let G be a group, i.e., a set provided with an internal operation, denoted by $\circ$, which is associative, admits a neutral element e and, for each element $x \in G$ there exists its inverse $x^{-1} \in G$ such that $x \circ x^{-1} = x^{-1} \circ x = e$.

G is said to be finitely generated if there exists a finite subset $H \subseteq G$ such that every $x \in G$ can be written as the composition of some elements of H, i.e., $x = h_1 \circ \cdots \circ h_k$ where $h_1, \ldots, h_k \in H$. The elements of H are called generators of G, while the composition chain $h_1 \circ \cdots \circ h_k$ is referred to as a decomposition of x.

Interestingly, a finitely generated group has a useful geometric interpretation that allows to connect our algebraic definition of the search space with the neighborhood structures frequently adopted to analyze combinatorial fitness landscapes (see for example [23, Ch. 5]). Indeed, given a group G with its finite subset of generators H, it is possible to represent the search space as a Cayley graph $\mathcal{C}(G, H)$, i.e., a directed graph whose vertices are the elements of G and, for any $x \in G$ and $h \in H$, the vertices x and $x \circ h$ are joined by a directed edge labeled with h. The Cayley graph is regular (every vertex has the same degree), strongly connected (for every ordered pair of vertices there is a path connecting them) and vertex-transitive (informally, it is not possible to recognize a vertex by simply looking at its incoming and outgoing edges). Moreover, the shortest path distance on $\mathcal{C}(G, H)$ induces a metric on G.

The properties of the Cayley graph allow to define a notion of difference in the search space. Indeed, for any two distinct solutions $x, y \in G$, there exists an oriented path in $\mathcal{C}(G, H)$ connecting y to x. This path is not unique in general, even if we restrict our attention to shortest paths. However, if $\langle h_1, \ldots, h_k \rangle$ is the sequence of the edge labels of any path from y to x, then its evaluation $h_1 \circ \cdots \circ h_k$ is always equal to $y^{-1} \circ x$. On the other hand, any sequence of edge labels $\langle h_1, \ldots, h_k \rangle$ can be applied to any vertex z obtaining a path from z to $z \circ h_1 \circ \cdots \circ h_k$. Hence, the sequences of edge labels can be thought as the discrete counterpart of the free vectors. Therefore, since every sequence of edges can be synthesized by its evaluation, we define the difference between two solutions $x, y \in G$ as

$$x \ominus y := y^{-1} \circ x. \tag{3}$$

Notably, since $x \ominus y \in G$, all the elements of G can be interpreted both as solutions and as differences between solutions, like in the Euclidean space.

The addition of $x, y \in G$ can be simply defined as

$$x \oplus y := x \circ y. \tag{4}$$

Interpreting x as a solution and y as the evaluation of a sequence of edges, then $x \circ y$ is exactly the solution reachable from x following a path with evaluation y. Note that the definitions of addition and subtraction are consistent to each other because $x = y \oplus (x \ominus y) = y \circ (y^{-1} \circ x) = x$ for any $x, y \in G$.

Given a solution $z \in G$, its magnitude $|z|$ is the length of a shortest path from e to z in $\mathcal{C}(G, H)$, or, equivalently, the length of a minimal decomposition of z using the generators in H. The latter interpretation allows to define the multiplication between a real number $F \in [0, 1]$ and an element $z \in G$ as the truncation of a minimal decomposition of z, i.e., given $z = h_1 \circ \cdots \circ h_{|z|}$,

$$F \odot z := h_1 \circ \cdots \circ h_k, \text{ where } k = \lceil F \cdot |z| \rceil. \tag{5}$$

Geometrically, $F \odot (x \ominus y)$ coincides with the truncation of a shortest path going from y to x in $\mathcal{C}(G, H)$, therefore, also this operator mimics its numerical counterpart in the Euclidean space. However, since the minimal decomposition is not unique in general, the multiplication operator is not uniquely defined. Nevertheless, it is possible to define a stochastic multiplication by using a random minimal decomposition of z. The process of random decomposition has to be as fair as possible to avoid biases in the DE algorithm, where the stochastic multiplication will be used.

```
1: function StochasticMultiplication(z ∈ G, F ∈ [0,1], H ⊆ G)
2:     s ← RandomizedDecomposer(z, H) ▷ s is a sequence of generators
3:     k ← ⌈F · Length(s)⌉
4:     r ← e                          ▷ e is the neutral element of G
5:     for i ← 1 to k do
6:         r ← r ∘ s[i]               ▷ ∘ is the composition operation of G
7:     end for
8:     return r                       ▷ r = F ⊙ z
9: end function
```

Fig. 2. Meta-Scheme of the Stochastic Multiplication Operator

The previous definitions (3), (4) and the algorithm of Fig. 2 allow to define the differential mutation scheme, outlined in equation (2), also in discrete search spaces whose solutions form a finitely generated group.

B. Application to the Permutations Space

The permutations of length n form a group, called symmetric group and denoted by $\mathcal{S}(n)$, in which the internal operation is the permutation composition $\circ$, i.e., the operator such that, for any $\pi_1, \pi_2 \in \mathcal{S}(n)$ and $1 \leq i \leq n$, $(\pi_1 \circ \pi_2)(i) = \pi_1(\pi_2(i))$. The neutral element of $\mathcal{S}(n)$ is the identity permutation e, and, for any $\pi \in \mathcal{S}(n)$, there exists $\pi^{-1} \in \mathcal{S}(n)$. Moreover, it is worthwhile to remark that we interpret permutations as orderings, i.e., $\pi(i)$ denotes the item (job) at position i in π, thus $\pi^{-1}(i)$ is the position of item (job) i in π.

In order to implement the stochastic multiplication in $\mathcal{S}(n)$, a set of generators has to be chosen. Different generating sets are possible and each one may lead to a different search space structure. Here, we consider the three main generating sets for permutations [10]:

- the set of all transpositions $T = \{(i, j)_T : 1 \leq i < j \leq n\}$, where $(i, j)_T$ denotes the permutation which swaps the items at places i and j;

- the set of all insertions $I = \{(i,j)_I : i \neq j \text{ and } 1 \leq i,j \leq n\}$, where $(i,j)_I$ denotes the permutation that shifts the item at place i to place j;
- the set of all simple transpositions $ST = \{(i,i{+}1)_T : 1 \leq i \leq n-1\}$, i.e., the permutations which swap two adjacent items (note that $(i,i{+}1)_T = (i,i{+}1)_I = (i{+}1,i)_I$).

T and I have a quadratic number of generators (respectively, $\binom{n}{2}$ and $(n-1)^2$) and both produce a search space diameter of $n-1$. Their induced search space structures are also known as, respectively, interchange and insertion neighborhoods, while the induced metrics are, respectively, the Cayley and Ulam distances [10]. ST, which is a proper subset of both T and I, has $n-1$ generators and a diameter of $\binom{n}{2}$. Its induced search space structure is also known as adjacent swap neighborhood, while the induced metric function d_{BS} is the bubble sort distance that counts the minimum number of adjacent swaps necessary to transform one permutation into the other [10].

Intuitively, a search space with a greater diameter and a smaller connectivity should reduce the arbitrarity in the computation of a random minimal decomposition, i.e., a shortest path in the Cayley graph. For this reason, we have decided to work with simple transpositions, i.e., with ST.

Furthermore, under the reasonable hypothesis that a smoother fitness landscape is a benefit for the search, the following experimental investigation has been made with the aim of verifying which generating set produces the smoother landscape on PFSP-TFT. We have randomly generated 120 problem instances (using the Taillard generator [24]), 10 instances for each $n \times m$ configuration, with $n \in \{10, 20, 50, 100\}$ and $m \in \{5, 10, 20\}$. Then, for each instance and for each generating set, we made 500 000 steps of random walk on the Cayley graph in order to compute the autocorrelation $\rho(1)$ among the fitness values of successively visited solutions. Also the correlation length $(\ln|\rho(1)|)^{-1}$ has been calculated. As described in [23, Ch. 5], high autocorrelation and correlation length values are an evidence that the analyzed landscape is smooth. The average values reported in Table I (best results in bold) show that the autocorrelations approach 1 when n increases and the correlation lengths are higher for ST, thus validating our choice.

TABLE I
AUTOCORRELATIONS AND CORRELATION LENGTHS

$n \times m$	Autocorr.			Corr. Lengths		
	ST	**T**	**I**	**ST**	**T**	**I**
10 × 5	**0.92**	0.74	0.81	**12.49**	3.26	4.77
10 × 10	**0.92**	0.73	0.81	**11.79**	3.25	4.68
10 × 20	**0.93**	0.74	0.82	**15.74**	3.40	5.03
20 × 5	**0.97**	0.86	0.91	**37.16**	6.52	10.63
20 × 10	**0.96**	0.85	0.90	**25.87**	6.31	9.79
20 × 20	**0.96**	0.85	0.90	**22.76**	6.15	9.41
50 × 5	**0.99**	0.94	0.96	**160.34**	15.74	27.54
50 × 10	**0.99**	0.93	0.96	**97.75**	14.20	24.01
50 × 20	**0.99**	0.93	0.96	**70.49**	14.16	22.94
100 × 5	**1.00**	0.97	0.98	**640.95**	29.88	53.67
100 × 10	**1.00**	0.96	0.98	**272.42**	26.92	47.28
100 × 20	**0.99**	0.96	0.98	**144.18**	24.46	41.60

Therefore, according to the meta-scheme reported in Fig. 2, in order to complete the design of the discrete differential mutation for $\mathcal{S}(n)$, we have to implement of a randomized decomposition algorithm that uses ST.

A minimal decomposition of any $\pi \in \mathcal{S}(n)$ in terms of simple transpositions can be obtained by ordering π through the classical bubble sort algorithm and by annotating, in the reverse order, the simple transpositions performed. Indeed, the sequence of simple transpositions $\langle \sigma_1, \ldots, \sigma_l \rangle$ performed by bubble sort is optimal in length and produces a decomposition of π^{-1}. By noting that the inverse of a simple transposition is itself, a decomposition of π is just the reversed sequence $\langle \sigma_l, \ldots, \sigma_1 \rangle$. The length l is equal to the number of inversions of π, i.e., the number of pairs (i,j) such that $i < j$ and $\pi(i) > \pi(j)$ [25]. Moreover, the distance $d_{BS}(\pi_1, \pi_2)$ of two permutations π_1, π_2 is equal to the number of inversions of $\pi_2^{-1} \circ \pi_1$ (or, for symmetry, of $\pi_1^{-1} \circ \pi_2$).

However, classical bubble sort is deterministic, i.e., fixing its input permutation, it produces always the same decomposition. In order to overcome this drawback, we propose the randomized version of bubble sort outlined in Fig. 3.

```
1: function RandBS(π ∈ S(n))
2:     s ← ⟨⟩
3:     Z ← {i : π(i) > π(i+1)}
4:     while Z ≠ ∅ do
5:         i ← get and remove a random element from Z
6:         Swap items π(i) and π(i+1) in π
7:         Append (i, i+1)_T to s
8:         if i > 0 and i − 1 ∉ Z and π(i−1) > π(i) then
9:             Insert i − 1 in Z
10:        end if
11:        if i < n − 1 and i + 1 ∉ Z and π(i+1) > π(i+2) then
12:            Insert i + 1 in Z
13:        end if
14:    end while
15:    Reverse the sequence s
16:    return s
17: end function
```

Fig. 3. Randomized Bubble Sort Algorithm

We now prove that RandBS produces a random minimal decomposition of π by sorting it. In the following, for the sake of clarity, we denote the original π to sort as π_0.

At every iteration, the set Z contains the first index of all the consecutive inversions of π (i.e., the inversions of the type $(i,i{+}1)$ which are also simple transpositions). Z is initialized in line 3. Then, at each iteration of the loop:

- a simple transposition $(i,i{+}1)_T$ is randomly chosen from Z and applied to π (lines 5 and 6), thus the number of inversion of π decreases by one;
- the sequence s is updated in line 7, thus s contains all the simple transpositions applied so far (by applying the simple transpositions in s to the right of π_0, the current π is obtained);
- Z is updated in the lines 8–13, by taking into account that the simple transposition $(i,i{+}1)_T$ can only generate the consecutive inversions $(i-1,i)$ and $(i+1,i+2)$.

At the end of the loop, $Z = \emptyset$, thus $\pi = e$ because it does not contain consecutive inversions. Finally, s is reversed in line 15. All these properties guarantee that:

- the length of s is optimal and equal to the number of inversions of π_0,

- the final content of s is a decomposition of π_0,
- the number of iterations is limited above by $\binom{n}{2} = O(n^2)$.

Moreover, since the computation of Z from scratch (line 3) costs $O(n)$, while each operation inside the loop can be implemented in $O(1)$, the overall time complexity of RandBS is $O(n^2)$ as its classical counterpart.

It is worthwhile to note that RandBS, when applied to π, through its iterations, performs a random walk on the subgraph of the Cayley graph (induced by ST) which includes: (i) the vertices represented by the permutations of the set $K(\pi) = \{\sigma \in \mathcal{S}(n) : d_{BS}(\pi, \sigma) + d_{BS}(\sigma, e) = d_{BS}(\pi, e)\}$, and (ii) a directed edge for any ordered pair (σ_1, σ_2) if and only if $\sigma_1, \sigma_2 \in K(\pi)$, (σ_1, σ_2) is already an edge in the Cayley graph and $d_{BS}(\sigma_1, e) = d_{BS}(\sigma_2, e) + 1$. Therefore, RandBS generates minimal decomposition sequences in a fair random way, although not with a uniform distribution. However, according to [26], a uniform distribution on the possible decompositions would require a so called "maximal entropy random walk", which has a much higher computational complexity.

Summarizing, the symmetric group, with its composition and inversion operations, and the randomized bubble sort algorithm, together with the abstract definitions provided in the previous section, allow to implement a discrete differential mutation for the permutations space as depicted in equation (2). Finally, it is worth to note that, since permutations composition and inversion cost both $O(n)$, the complexity of the differential mutation is dominated by the randomized bubble sort complexity, i.e., $O(n^2)$.

For the sake of clarity we provide an illustrative example of how the differential mutation works. Let $n = 5$, $F = 0.5$, and the given permutations be $\pi_0 = \langle 3, 4, 1, 2, 5\rangle$, $\pi_1 = \langle 1, 4, 2, 5, 3\rangle$ and $\pi_2 = \langle 5, 3, 1, 4, 2\rangle$. In order to compute the mutant permutation $\nu = \pi_0 \oplus F \odot (\pi_1 \ominus \pi_2)$, we have to first calculate $\pi_1 \ominus \pi_2 = \pi_2^{-1} \circ \pi_2$. Since $\pi_2^{-1} = \langle 3, 5, 2, 4, 1\rangle$, then $\pi_2^{-1} \circ \pi_1 = \langle 3, 4, 5, 1, 2\rangle$. Now, a possible result of RandBS$(\pi_1 \ominus \pi_2)$ is the decomposition $\langle (2,3)_T, (3,4)_T, (1,2)_T, (2,3)_T, (4,5)_T, (3,4)_T\rangle$ which has length 6. Therefore, the truncation $0.5 \odot (\pi_1 \ominus \pi_2)$ is $(2,3)_T \circ (3,4)_T \circ (1,2)_T = \langle 3, 1, 4, 2, 5\rangle$. Finally, by composing π_0 with the truncated difference, we obtain $\nu = \langle 1, 3, 2, 4, 5\rangle$ and, as expected, $d_{BS}(\nu, \pi_0) = 3$.

V. Other DEP Components

In the following we provide the description of the other algorithmic components outlined in the DEP scheme reported in Fig. 1.

A. Initialization

Two initialization schemes have been considered: a completely random (R_INIT) and a heuristic based (H_INIT) initialization. In R_INIT, N permutations are randomly generated. Instead, H_INIT builds one solution using the constructive heuristic $LR(n/m)$ [9], while it randomly generates the other $N - 1$ permutations.

B. Crossover

The crossover between the population individual π_i and the mutant ν_i is performed according to the two-point crossover version II (TPII) described in [27] (and also used by AGA [15]).

TPII produces the two offspring individuals υ_i' and υ_i'' as follows. First, two cut-points j, k, such that $1 \leq j \leq k \leq n$, are randomly generated. Then, $\upsilon_i'(h) \leftarrow \pi_i(h)$ for $j \leq h \leq k$, while the missing jobs are inserted in υ_i' starting from the leftmost free place and following the order of their appearance in ν_i. The other offspring υ_i'' is filled in the same way but by reversing the role of π_i and ν_i.

For instance, TPII applied to $\pi = \langle 1, 2, 4, 5|3, 9, 8|7, 6\rangle$ and $\nu = \langle 3, 7, 6, 5|1, 4, 2|9, 8\rangle$, using the indicated cut-points, produces $\upsilon' = \langle 7, 6, 5, 1|3, 9, 8|4, 2\rangle$ and $\upsilon'' = \langle 5, 3, 9, 8|1, 4, 2|7, 6\rangle$.

TPII is parameterless and its computational complexity is $O(n)$. Furthermore, it is worth to note that TPII does not work in the same metric space of the differential mutation operator. Indeed, it is possible to prove that it is not geometric (in the sense of [28]) under the distance d_{BS}[1]. Actually, this is in accordance with classical DE, where the differential mutation works basing on the Euclidean distance, while the binomial crossover is not geometric under this distance.

C. θ-Selection

Since the crossover generates two offspring solutions, the selection is performed in two stages.

First, the trial solution υ_i is the fittest between the two offsprings υ_i' and υ_i''.

Then, the new population individual π_i' is chosen by means of a biased selection scheme, called θ-selection, between υ_i and π_i according to

$$\pi_i' \leftarrow \begin{cases} \upsilon_i & \text{if } f(\upsilon_i) < f(\pi_i) \text{ or } r < \max\{\theta - \Delta_i, 0\} \\ \pi_i & \text{otherwise} \end{cases} \quad (6)$$

where $\Delta_i = (f(\upsilon_i) - f(\pi_i))/f(\pi_i)$ is the relative fitness variation of υ_i with respect to π_i, r is a random number in $[0, 1]$, and $\theta \in [0, 1]$ is a selection parameter.

Similarly to classical DE selection, υ_i enters the next generation population if it is fitter than π_i. Otherwise, a slightly worsening υ_i may be selected with a small probability that linearly shades from θ, when $\Delta_i = 0$, to 0, when $\Delta_i = \theta$. Therefore, θ regulates the selective pressure and allows to prevent the stagnation of the search. Finally, note that the θ-selection is a generalization of classical DE selection that is reproduced by setting $\theta = 0$.

D. Restart

When all the population individuals are the same, the DE genetic operators (also in the classical continuous DE) are not

[1]This can be shown using the TPII example above. Indeed, since $d_{BS}(\pi, \upsilon') + d_{BS}(\upsilon', \nu) = 24 + 8 \neq 20 = d_{BS}(\pi, \nu)$, the first offspring υ' is not in the metric segment, under the distance d_{BS}, between its parents π and ν. The same is true also for the second offspring υ''.

able to evolve the population anymore. To address this issue, a restart procedure has been introduced.

In DEP, a restart is triggered when all the population fitnesses are the same. Indeed, it has been experimentally observed that, in more than the 99% of the cases, this is equivalent to check for the presence of a single genotype, but it is much more efficient.

When a restart is triggered, one population individual is kept, while the others are randomly reinitialized.

E. Local Search

After every restart, a local search procedure is optionally applied to the best population individual. Three local search application schemes are considered: N_LS, B_LS, and L_LS. N_LS does not perform any local search. B_LS performs a Baldwinian local search, i.e., the (hopefully) improved solution is recorded but does not enter the DEP population. L_LS performs a Lamarckian local search, i.e., the (hopefully) improved solution replaces the original population solution. With respect to N_LS, both B_LS and L_LS consume fitness evaluations. However, B_LS, conversely from L_LS, does not influence the DEP evolution dynamics.

The local search procedure employed is similar to VNS_4 [14], though we do not use any perturbation scheme. A first-improvement local search using the interchange neighbourhood is carried out until a local minimum is found. Then, the best solution in its insertion neighbourhood is chosen and the whole process is iterated again until a local minimum common to both neighbourhoods is reached.

VI. Experimental Analysis

The experimental analysis has been performed with four purposes: calibrate the DEP parameters, compare DEP with the state-of-the-art PFSP-TFT algorithms, study the impact of the different DEP components, and analyze the computational times.

It is worth to note that, while the comparison analysis has been held using the standard benchmark suites proposed in [24] and [3], the calibration instances have been randomly generated by using the same generator. This avoids an over-tuning of the DEP parameters to the test instances and it performs a calibration oriented to the most widely accepted benchmark suites. The dimension of the benchmark instances varies with $20 \leq n \leq 500$ and $m \in \{5, 10, 20\}$.

In order to be consistent with the other works on this subject (e.g., [3], [11], [15]), the performance measure employed is the average relative percentage deviation (ARPD) defined as

$$ARPD = \frac{1}{k} \sum_{i=1}^{k} \frac{(Alg_i - Best) \times 100}{Best} \tag{7}$$

where Alg_i is the final TFT value found by the algorithm Alg in its i-th run of a total of k runs, while $Best$ is a reference TFT value for the problem at hand. Moreover, the ARPDs obtained by the algorithms have been compared by means of non-parametric statistical tests [29].

Finally, the maximum numbers of fitness evaluations are the same reported in [3, Tables III and IV].

A. Calibration of DEP

DEP has five parameters: the scale factor F, the population size N, the selection parameter θ, the initialization scheme, and the local search application scheme. Note that, differently from classical DE, DEP has no crossover parameter.

The popular self-adaptive scheme proposed in [30] has been used to dynamically adapt F during the evolution. The other four parameters have been chosen by means of a full factorial experimental analysis considering the following levels:

- N: 50, 100, and 200;
- θ: 0, 0.005, 0.01, and 0.02;
- initialization: R_INIT, and H_INIT;
- local search application: N_LS, B_LS, and L_LS.

Therefore, we have a total of $3 \times 4 \times 2 \times 3 = 72$ DEP settings to compare.

The experimental design has been modeled with the aim of providing one single DEP setting that (hopefully) performs well across the different problem configurations. The candidate DEP settings were compared on 11 problem configurations, all the $n \times m$ combinations with $n \in \{20, 50, 100, 200\}$ and $m \in \{5, 10, 20\}$ except 200×5. For each problem, 10 instances were randomly generated and every DEP setting was run once for each instance, thus having a total of $72 \times 11 \times 10 = 7\,920$ runs. Then, an ARPD value is produced for every setting and every problem configuration. According to [31, Th. 4.12], this experimental setting minimizes the variance of the averaged performance estimator ARPD.

Table II provides, for every problem configuration, the best performing DEP setting together with its ARPD on the problem and its overall ARPD. Since all the DEP settings perform exactly the same on the 20 jobs instances, these are not reported.

TABLE II
Calibration Results

$n \times m$	Best DEP Setting on $n \times m$				Problem	Overall
	N	θ	Init.	LS	ARPD	ARPD
50×5	200	0.005	R_INIT	N_LS	0.05	1.58
50×10	100	0.01	R_INIT	L_LS	0.14	0.17
50×20	50	0.01	H_INIT	B_LS	0.08	0.26
100×5	100	0.01	H_INIT	L_LS	0.03	0.17
100×10	100	0.01	R_INIT	N_LS	0.12	0.17
100×20	50	0.01	H_INIT	B_LS	0.08	0.26
200×10	200	0.01	R_INIT	L_LS	0.04	0.57
200×20	**100**	**0.01**	**H_INIT**	**B_LS**	0.12	**0.15**

Interestingly, the best performing setting on 200×20, i.e., $(N = 100, \theta = 0.01, \text{H\_INIT}, \text{B\_LS})$, is also the setting with the best overall ARPD. A statistical analysis has been also performed in order to validate the experimental results of the seven settings of Table II (note that the best settings on 50×20 and 100×20 coincide). Due to the different dimensionalities of the problems considered, we have employed the non-parametric Quade test that, as suggested in [29], allows to take into account the problems difficulties by performing a weighted ranking analysis based on the intra-problem ARPD variabilities. The Quade test shows that the best setting in terms of overall ARPD has also the best Quade average rank. Furthermore, the Quade p-value of 0.004 indicates the

existence of statistical differences among the performances of the analyzed settings. Thus, the Finner post-hoc procedure [29] has been applied in order to compare the selected settings. The obtained p-values show, with a high evidence, that the four settings with $N = 100$ and $\theta = 0.01$ perform statistically the same and outperform the others.

Therefore, the setting selected for further investigations is $(N = 100, \theta = 0.01, \text{H\_INIT}, \text{B\_LS})$.

B. Comparison with state-of-the-art algorithms

The selected DEP setting has been experimentally compared with six state-of-the-art PFSP-TFT algorithms: the three local search schemes ILS, IGA and VNS_4, the evolutionary algorithm GM-EDA, and the two hybrid evolutionary schemes AGA and HGM-EDA (all of them are described in Section II).

The seven algorithms have been evaluated on the well known 120 benchmark instances proposed by Taillard in [24]. Consistently with [3], for each algorithm, 20 runs per instance were allocated using the evaluations budgets of [3]. Furthermore, in order to perform a fair comparison, the results for AGA, VNS_4, GM-EDA and HGM-EDA have been directly taken from the supplementary data of [3], while ILS and IGA have been executed with an equal number of fitness evaluations and using the settings suggested in their original paper [11].

In order to detect the statistical differences among the ARPD results, as suggested in [29], the non-parametric Friedman test and the Finner post-hoc procedure have been applied in every $n \times m$ problem configuration. The significance level has been set to $\alpha = 0.05$.

For each problem instance, the best TFT value and the ARPDs of each algorithm are provided in Table III. The best ARPD on each instance is reported in bold. The results marked with the asterisk denote that the best TFT value has been achieved at least once among the 20 runs of the algorithm. The TFTs in bold indicates a new best known value with respect to both [3] and [11]. For each problem configuration and for each algorithm, Table III also shows the Friedman average rank together with a symbol indicating the result of the statistical comparison with DEP: "=" if the performances are statistically equal, and "−" or "+" if, respectively, DEP statistically outperforms or it is outperformed by the comparing algorithm. Lastly, the ARPDs aggregated for every problem configuration, and the overall ARPDs, are shown in Fig. 4.

The experimental results show that, in 75 instances over 120, DEP obtains the best TFT, and, most remarkably, 14 of them are new best known solutions on the 20 instances of the $200 \times m$ problems which, for their size, are reputed to be difficult. The robustness of DEP is proved by the fact that it has the lowest ARPD results in 95 instances over 120 (almost the 80% of the benchmark suite). Note also that, in almost all the problems with 100 and 200 jobs, DEP is the best algorithm on average. DEP has also the best Friedman average ranks on 10 over 12 problem configurations and, as shown in Fig. 4, its aggregated ARPDs are the best ones or they are very close to the best ones.

These results can be summarized as follows.

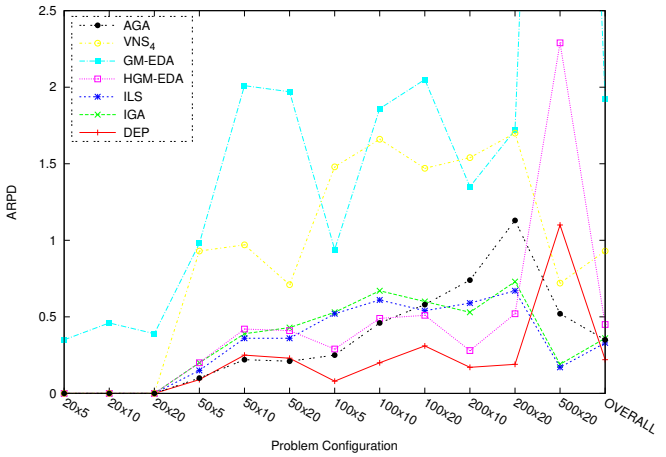

Fig. 4. Per-problem comparison and overall results

- In the $20 \times m$ problems, all the algorithms perform the same, except GM-EDA which is significantly worse than all the others.
- For the problems with 50 jobs, DEP has only one comparable competitor, i.e., AGA, while all the other algorithms are significantly worse.
- In the $100 \times m$ problems, DEP has average rank values not greater than 1.1 and it significantly outperforms every other competitor.
- A similar behavior is found for the problems with 200 jobs, where no competitor is able to match the performances of DEP.
- The only weakness is found in the problems with 500 jobs where DEP is outperformed by AGA and the local search schemes. A plausible reason is the slow convergence of DEP in these cases. Indeed, zero or almost zero restarts have been observed on the 500 jobs instances. However, its aggregated ARPD is not too far from the best ones if compared to those of GM-EDA and HGM-EDA which work in a similar metric space.
- Finally, and most notably, DEP reports the best overall ARPD.

In conclusion, DEP can be considered among the state-of-the-art PFSP-TFT algorithms. Note also that 9 new best known solutions have been obtained by our runs of ILS and IGA (one on the first 200×20 instance and the others on the 500×20 instances). This is plausibly due to the larger number of fitness evaluations with respect to [11].

C. Additional Experiments

The previous analysis shows that DEP performances are excellent for $n \leq 200$, while they degrade when $n = 500$. Since the Taillard benchmark suite [24] does not consider problem instances with $200 < n < 500$, a further experimental analysis has been performed on the additional 100 benchmark instances, with $n \in \{250, 300, 350, 400, 450\}$ and $m \in \{10, 20\}$, proposed in [3]. These additional PFSP instances were randomly generated using the same generator proposed by Taillard [24], thus constituting a consistent extension of the Taillard benchmark suite.

TABLE III
EXPERIMENTAL COMPARISON OF DEP WRT STATE-OF-THE-ART PFSP-TFT ALGORITHMS ON TAILLARD BENCHMARKS

Problem Instance	Best TFT	AGA	VNS_4	GM EDA	HGM EDA	ILS	IGA	DEP
20 × 5	14 033	**0.00** *	**0.00** *	0.18 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	15 151	**0.00** *	**0.00** *	0.48	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	13 301	**0.00** *	**0.00** *	0.50 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	15 447	**0.00** *	**0.00** *	0.43 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	13 529	**0.00** *	**0.00** *	0.21 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	13 123	**0.00** *	**0.00** *	0.08 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	13 548	**0.00** *	**0.00** *	0.79	**0.00** *	**0.00** *	0.02 *	**0.00** *
	13 948	**0.00** *	**0.00** *	0.18 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	14 295	**0.00** *	**0.00** *	0.18 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	12 943	**0.00** *	**0.00** *	0.46 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
Friedman AvgRank		**3.45**	**3.45**	7.0	**3.45**	**3.45**	3.75	**3.45**
Stat.Comp. vs DEP		=	=	−	=	=	=	
20 × 10	20 911	**0.00** *	**0.00** *	0.45 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	22 440	**0.00** *	**0.00** *	0.54 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	19 833	**0.00** *	**0.00** *	0.31 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	18 710	**0.00** *	**0.00** *	0.75	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	18 641	**0.00** *	**0.00** *	0.35	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	19 245	**0.00** *	**0.00** *	0.77	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	18 363	**0.00** *	**0.00** *	0.47 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	20 241	**0.00** *	**0.00** *	0.47 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	20 330	**0.00** *	**0.00** *	0.27 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	21 320	**0.00** *	**0.00** *	0.24 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
Friedman AvgRank		**3.5**	**3.5**	7.0	**3.5**	**3.5**	**3.5**	**3.5**
Stat.Comp. vs DEP		=	=	−	=	=	=	
20 × 20	33 623	**0.00** *	**0.00** *	0.65 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	31 587	**0.00** *	**0.00** *	0.28 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	33 920	**0.00** *	**0.00** *	0.04 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	31 661	**0.00** *	**0.00** *	0.28 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	34 557	**0.00** *	**0.00** *	0.26	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	32 564	**0.00** *	**0.00** *	0.30 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	32 922	**0.00** *	**0.00** *	0.61	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	32 412	**0.00** *	**0.00** *	0.52	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	33 600	**0.00** *	**0.00** *	0.56 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
	32 262	**0.00** *	**0.00** *	0.41 *	**0.00** *	**0.00** *	**0.00** *	**0.00** *
Friedman AvgRank		**3.5**	**3.5**	7.0	**3.5**	**3.5**	**3.5**	**3.5**
Stat.Comp. vs DEP		=	=	−	=	=	=	
50 × 5	64 803	**0.05** *	0.78	0.79	0.12 *	**0.05** *	0.07	**0.05**
	68 062	**0.06** *	0.88	0.94	0.12	0.13	0.14	0.08
	63 162	**0.19** *	1.21	1.34	0.38	0.24 *	0.24	0.21
	68 226	0.17	1.12	1.27	0.22 *	**0.13**	0.26 *	**0.13**
	69 392	**0.09** *	0.87	0.89	0.15	0.16	0.14	**0.09**
	66 841	0.10 *	0.80	0.82	0.18 *	0.19	0.16 *	**0.04**
	66 253	**0.03**	0.74	0.96	0.08	0.09 *	0.10 *	**0.03**
	64 359	**0.05**	0.89	0.97	0.23 *	0.15	0.32	**0.05** *
	62 981	0.09 *	0.83	0.81	0.14 *	0.19	0.23 *	**0.05**
	68 811	0.20	1.18	1.06	0.34	0.22	0.33 *	**0.15**
Friedman AvgRank		1.7	6.2	6.8	4.0	3.5	4.3	**1.5**
Stat.Comp. vs DEP		=	−	−	−	−	−	
50 × 10	87 204	0.33	1.12	2.11	0.39	0.44	0.40 *	**0.18** *
	82 820	**0.22** *	1.09	2.45	0.60 *	0.59	0.39 *	0.30
	79 987	0.23 *	1.07	1.84	0.36	0.32	0.40	**0.22**
	86 545	0.21	0.94	1.87	0.36	0.26	0.31	**0.16** *
	86 424	**0.17**	0.93	2.05	0.41	0.24	0.35 *	0.28
	86 637	0.13 *	0.77	1.55	0.29	0.17	0.35	**0.11**
	88 866	**0.25** *	0.89	1.97	0.48	0.28	0.27	0.42
	86 820	0.19	0.95	2.04	0.36	0.43	0.44	**0.01** *
	85 526	0.29	1.11	2.10	0.42 *	0.44 *	0.49	**0.28**
	87 998	**0.18**	0.85	2.09	0.54	0.39 *	0.45	0.51
Friedman AvgRank		**1.6**	6.0	7.0	4.2	3.3	4.0	1.9
Stat.Comp. vs DEP		=	−	−	−	−	−	
50 × 20	125 831	**0.10** *	0.65	1.76	0.39 *	0.33 *	0.41	0.14
	119 247	**0.05**	0.52	1.59	0.23	0.14	0.15 *	0.07
	116 459	**0.19** *	0.73	2.24	0.44 *	0.33	0.36	0.28 *
	120 712	**0.22** *	0.61	1.92	0.34	0.35	0.36	0.34
	118 184	0.40	0.86	2.30	0.52 *	0.48	0.60	**0.39**
	120 703	0.19 *	0.62	1.78	0.35	0.34	0.47	**0.16**
	122 962	0.38	0.71	2.10	0.47 *	0.52	0.47	**0.36**
	122 489	0.16	0.75	2.24	0.55 *	0.41	0.45	**0.14**
	121 872	0.16 *	0.76	1.79	0.37 *	0.34	0.41	**0.12** *
	124 064	**0.23** *	0.90	1.95	0.42	0.35	0.64	0.29
Friedman AvgRank		**1.5**	6.0	7.0	4.1	3.3	4.6	**1.5**
Stat.Comp. vs DEP		=	−	−	−	−	−	

Problem Instance	Best TFT	AGA	VNS_4	GM EDA	HGM EDA	ILS	IGA	DEP
100 × 5	253 605	0.29	1.25	0.87	0.23	0.48	0.46	**0.05** *
	242 579	0.30	1.80	1.08	0.35	0.80	0.82	**0.05** *
	238 075	0.22	1.49	0.85	0.26	0.34	0.29	**0.07** *
	227 889	0.17 *	1.29	0.78	0.20	0.45	0.32	**0.06**
	240 589	0.21	1.29	0.80	0.23 *	0.56	0.45	**0.02**
	232 689	0.32	1.52	0.90	0.28	0.37	0.47	**0.06** *
	240 750	**0.11** *	1.31	0.97	0.31	0.43	0.30	0.22
	231 064	0.29	1.79	1.06	0.35	0.56	0.81	**0.07** *
	248 039	0.40	1.66	1.05	0.38	0.66	0.67	**0.09** *
	243 258	0.19	1.44	1.00	0.28	0.51	0.70	**0.07** *
Friedman AvgRank		2.2	7.0	6.0	2.8	4.5	4.4	**1.1**
Stat.Comp. vs DEP		−	−	−	−	−	−	
100 × 10	299 101	0.43	1.63	1.80	0.44	0.42	0.56	**0.16** *
	274 566	0.60	1.58	2.08	0.69	0.54	0.85	**0.28** *
	288 543	0.37	1.57	1.74	0.38	0.82	0.88	**0.18** *
	301 552	0.50	1.79	2.08	0.53	0.88	0.96	**0.18** *
	284 722	0.61	1.64	1.95	0.54	0.73	0.64	**0.22** *
	270 483	0.42	1.76	1.83	0.45	0.52	0.63	**0.19** *
	280 257	0.37	1.58	1.65	0.40	0.46	0.41	**0.25** *
	291 231	0.49	1.77	2.03	0.61	0.69	0.72	**0.27** *
	302 624	0.36 *	1.46	1.76	0.41	0.57	0.66	**0.20**
	291 705	0.48	1.84	1.68	0.50	0.47	0.45	**0.06** *
Friedman AvgRank		2.5	6.1	6.9	3.3	3.7	4.5	**1.0**
Stat.Comp. vs DEP		−	−	−	−	−	−	
100 × 20	366 438	0.80	1.70	2.26	0.67	0.79	0.71	**0.37** *
	373 138	0.55	1.43	2.04	0.58	0.54	0.73	**0.25** *
	371 206	0.53	1.37	1.99	0.41	0.38 *	0.49	**0.27**
	373 574	0.60	1.36	1.92	0.45	0.50	0.49	**0.26** *
	369 850	0.59	1.36	1.93	0.48	0.57 *	0.50	**0.21**
	372 752	0.51	1.46	2.17	0.42	0.56	0.56	**0.30** *
	373 447	0.70	1.82	2.19	0.63	0.73	0.90	**0.33** *
	385 456	0.46	1.41	1.96	0.43 *	0.48	0.64	**0.20**
	375 352	0.62	1.52	2.01	0.52	0.44	0.54	**0.41** *
	379 899	0.48	1.29	2.05	0.49 *	**0.44**	0.48	0.46
Friedman AvgRank		4.1	6.0	7.0	2.7	3.1	4.0	**1.1**
Stat.Comp. vs DEP		−	−	−	−	−	−	
200 × 10	1 047 541	0.49	1.26	1.20	0.19	0.29	**0.15** *	0.22
	1 035 783	0.94	1.54	1.49	0.32	0.71	0.65	**0.15** *
	1 045 706	0.66	1.62	1.30	0.32	0.55	0.48	**0.15** *
	1 029 580	0.77	1.65	1.38	0.45	0.51	0.48	**0.12** *
	1 036 464	0.68	1.35	1.37	0.19	0.42	0.39	**0.13** *
	1 006 650	0.50	1.36	1.39	**0.19** *	0.67	0.55	0.23
	1 052 786	0.95	1.66	1.23	0.24	0.61	0.58	**0.10** *
	1 044 961	0.62	1.51	1.39	0.25	0.35	0.34	**0.11** *
	1 023 315	0.81	1.61	1.29	0.28	0.93	0.86	**0.24** *
	1 029 198	0.97	1.87	1.48	0.39	0.87	0.78	**0.25** *
Friedman AvgRank		4.6	6.8	6.2	1.9	4.2	3.0	**1.3**
Stat.Comp. vs DEP		−	−	−	−	−	−	
200 × 20	**1 225 282**	0.76	1.48	1.72	0.39	0.30 *	0.40	**0.20**
	1 239 246	1.07	1.67	1.66	0.54	0.64	0.74	**0.21** *
	1 263 134	1.08	1.65	1.57	0.48	0.44	0.51	**0.26** *
	1 233 443	1.25	1.84	1.73	0.58	0.83	1.03	**0.24** *
	1 220 117	1.12	1.79	1.93	0.53	0.80	0.78	**0.17** *
	1 223 238	1.17	1.69	1.69	0.46	0.77	0.86	**0.19** *
	1 237 116	1.03	1.65	1.66	0.64	0.70	0.81	**0.15** *
	1 238 975	1.25	1.72	1.72	0.51	0.81	0.72	**0.19** *
	1 225 186	1.44	1.91	1.80	0.59	0.88	0.87	**0.14** *
	1 244 200	1.16	1.62	1.68	0.52	0.48	0.55	**0.11** *
Friedman AvgRank		5.0	6.5	6.5	2.3	3.0	3.7	**1.0**
Stat.Comp. vs DEP		−	−	−	−	−	−	
500 × 20	**6 687 476**	0.42	0.66	9.24	2.34	**0.13** *	0.22	1.32
	6 809 182	0.55	0.69	8.91	2.25	**0.24** *	0.36	0.96
	6 734 895	0.42	0.59	8.66	2.23	0.14	**0.13** *	1.25
	6 767 341	0.56	0.74	9.07	2.18	**0.16** *	0.19	1.13
	6 720 679	0.91	0.93	9.28	2.45	**0.15** *	0.17	1.26
	6 723 390	0.60	0.84	9.04	2.56	**0.14**	0.18 *	0.73
	6 681 650	0.68	0.86	9.59	2.46	**0.17** *	0.21	1.34
	6 769 821	0.31 *	0.58	8.62	2.09	0.28	**0.20**	0.73
	6 701 696	0.43	0.74	8.99	2.19	**0.10** *	0.12	1.24
	6 755 620	0.37	0.62	8.70	2.18	0.15	**0.10** *	1.04
Friedman AvgRank		3.0	4.1	7.0	6.0	**1.3**	1.7	4.9
Stat.Comp. vs DEP		+	+	−	−	+	+	

Experiments have been held using the same setting of Section VI-B. The only exception is the exclusion of the worst performing algorithm, i.e., GM-EDA.

Fig. 5 and 6 show the aggregated ARPDs for each problem configuration, respectively, for $m = 10$ and $m = 20$. These ARPD values, the Friedman average ranks and the results of the statistical comparison are reported in Tables IV and V. More detailed results are provided on the web as supplementary data [32].

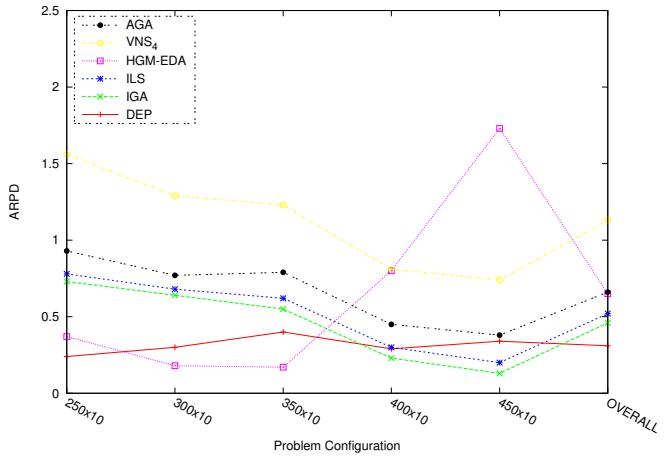

Fig. 5. Aggregated ARPDs on the additional problems with $m = 10$

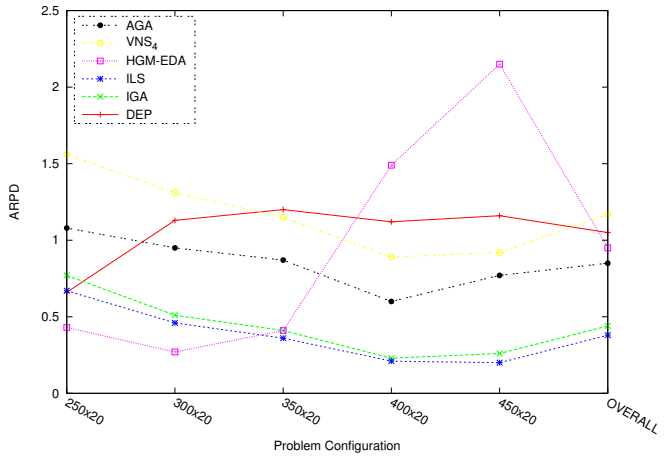

Fig. 6. Aggregated ARPDs on the additional problems with $m = 20$

As shown by Fig. 5 and Table IV, on the problems with 10 machines, DEP obtains the best overall ARPD and it is the only algorithm which persistently shows good performances across the various values of n. Indeed, while HGM-EDA performances degrade when n increases, the other schemes based on local search procedures, as expected, become competitive for larger values of n.

On the other hand, as shown by Fig. 6 and Table V, when $m = 20$ and $n \geq 300$, the ARPDs of DEP are far from those of the best algorithm. However, the behavior of DEP performances looks to be approximately constant with the increasing of n, as for the instances with $m = 10$. Therefore, regarding the overall ARPD on the $n \times 20$ problems, DEP, although worse than IGA and ILS, is still comparable with HGM-EDA, AGA, and VNS$_4$.

TABLE IV
EXPERIMENTAL COMPARISON OF DEP WRT STATE-OF-THE-ART PFSP-TFT ALGORITHMS ON THE ADDITIONAL BENCHMARKS ($m = 10$)

$n \times m$	AGA	VNS$_4$	HGM EDA	ILS	IGA	DEP
250 × 10	0.93	1.56	0.37	0.78	0.73	**0.24**
AvgRank	4.4	6.0	1.9	4.0	3.2	**1.5**
Stat.Comp.	−	−	=	−	−	
300 × 10	0.77	1.29	**0.18**	0.68	0.64	0.30
AvgRank	4.6	6.0	**1.2**	4.0	3.4	1.8
Stat.Comp.	−	−	+	−	−	
350 × 10	0.79	1.23	**0.17**	0.62	0.55	0.40
AvgRank	4.6	6.0	**1.2**	4.3	2.9	2.0
Stat.Comp.	−	−	+	−	−	
400 × 10	0.45	0.81	0.80	0.30	**0.23**	0.29
AvgRank	3.6	5.3	5.5	2.7	**1.6**	2.3
Stat.Comp.	−	−	−	=	=	
450 × 10	0.38	0.74	1.73	0.20	**0.13**	0.34
AvgRank	3.4	5.0	6.0	2.2	**1.1**	3.3
Stat.Comp.	=	−	−	+	+	
OVERALL ARPD	0.66	1.13	0.65	0.52	0.46	**0.31**

TABLE V
EXPERIMENTAL COMPARISON OF DEP WRT STATE-OF-THE-ART PFSP-TFT ALGORITHMS ON THE ADDITIONAL BENCHMARKS ($m = 20$)

$n \times m$	AGA	VNS$_4$	HGM EDA	ILS	IGA	DEP
250 × 20	1.08	1.56	**0.43**	0.67	0.77	0.66
AvgRank	5.0	6.0	**1.2**	2.7	3.5	2.6
Stat.Comp.	−	−	+	=	−	
300 × 20	0.95	1.31	**0.27**	0.46	0.51	1.13
AvgRank	4.1	5.9	**1.4**	2.2	2.4	5.0
Stat.Comp.	+	−	+	+	+	
350 × 20	0.87	1.15	0.41	**0.36**	0.41	1.20
AvgRank	4.1	5.3	2.0	**1.6**	2.4	5.6
Stat.Comp.	+	=	+	+	+	
400 × 20	0.60	0.89	1.49	**0.21**	0.23	1.12
AvgRank	3.0	4.1	6.0	**1.5**	**1.5**	4.9
Stat.Comp.	+	+	−	+	+	
450 × 20	0.77	0.92	2.15	**0.20**	0.26	1.16
AvgRank	3.2	3.8	6.0	**1.2**	1.8	5.0
Stat.Comp.	+	+	−	+	+	
OVERALL ARPD	0.85	1.17	0.95	**0.38**	0.44	1.05

This experimental session shows that DEP performances are excellent and apparently independent of m when $n \leq 250$, while, for $n \geq 300$ and conversely from the other competitors, they are heavily influenced by the number of machines. In order to investigate this phenomenon we have analyzed the average number of restarts performed by DEP on the various problem configurations. This quantity clearly relates with the convergence speed of the DEP evolution. The greater is the number of observed restarts, the faster is the DEP evolution and vice versa. The analysis shows that: (i) the number of restarts gracefully decreases when n increases, (ii) fixing n, the number of observed restarts is considerably smaller for $m = 20$ than for $m = 10$. Therefore, the slow convergence looks to be a plausible explanation of DEP performances on the problems with $m = 20$. This suggests that the number of machines affects the characteristics of the landscape navigated by DEP.

Finally, further experiments have been held in order to compare DEP and the version of HGM-EDA with improved initialization, namely Guided HGM-EDA [3]. Similarly to [3],

the experimental comparison has been performed on the first instance of the four largest problem configurations: 400×20, 450×10, 450×20, and 500×20. The experiments clearly show that, although Guided HGM-EDA improves the results of HGM-EDA, these are still worse than DEP results (the detailed data for this experiment are provided on the web as supplementary data [32]).

D. DEP Components Analysis

A study of the various DEP algorithmic components has been mainly performed by analyzing the experimental results obtained in the calibration phase (see Section VI-A). The analyses of each DEP component are discussed in the following, while a full detail is provided as supplementary data on the web [32].

1) Initialization: The results for H_INIT and R_INIT, averaged over all the calibration runs and aggregated for every problem configuration, show that the heuristic-based initialization outperforms the pure random scheme, though they are comparable for small problems.

2) Mutation: The analysis of the mutation operator aims at verifying (i) if the discrete differential mutation is effectively better than a classical random mutation, and (ii) if the randomization of bubble sort is necessary.

In order to investigate these points, five modified DEP schemes have been devised, each one adopting a different mutation operator:

- RandBS_MUT: the original DEP differential mutation (see Section IV),
- R_MUT: it generates a completely random mutant,
- Q_MUT: it computes the bubble sort distance d between the two individuals π_{r_1} and π_{r_2}, then it applies $F \cdot d$ random adjacent swaps to the base individual π_{r_0},
- ClassicBS_MUT: it adopts the classical bubble sort,
- CocktailBS_MUT: it employs the variant of bubble sort known as cocktail sort (it alternates left-to-right and right-to-left scans during the sorting process).

Q_MUT is a simplified version of RandBS_MUT because it uses only the distance d between two elements without considering their difference, thus the perturbation strength of both methods is the same. All the other components have been set as described in Section VI-A and the analysis has been performed using the calibration instances with $n \leq 100$. The aggregated and overall results are depicted in Fig. 7.

The graph shows that, although all the mutations perform the same on the 20 jobs problems, RandBS_MUT clearly outperforms the other schemes on the larger problems. Therefore, both the hypotheses are experimentally validated. Interestingly, the Q_MUT performances are in the middle between RandBS_MUT and R_MUT. This suggests that both magnitude and direction are important to guide the search. Furthermore, ClassicBS_MUT and CocktailBS_MUT performs similarly and both are worse than Q_MUT. This confirms the hypothesis that a randomized navigation is definitely necessary.

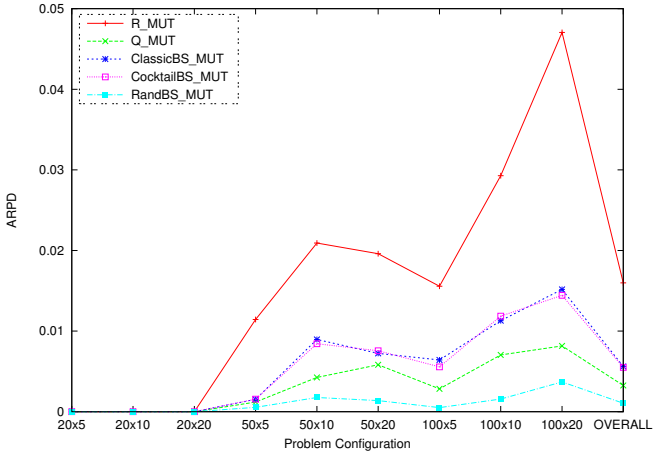

Fig. 7. Comparison of DEP using different mutation schemes

3) Crossover: As described in Section V-B, the crossover adopted in DEP produces the two offsprings υ' and υ'' from which, after a preliminary selection, the trial solution υ is chosen. This scheme doubles the number of fitness evaluations per generation and halves the number of generations needed to reach the evaluations cap. Nevertheless, the double offspring crossover has a significantly larger success rate than any possible version with a single offspring. Indeed, the conducted experimental analysis shows that the choice of only one offspring, without considering the other, consistently deteriorates the probability of replacing the original population individual, thus slowing down the evolution.

4) Selection: The θ-selection scheme introduced in DEP allows to regulate the acceptance threshold for the trial solution. The four θ values adopted in the calibration phase, averaged over all the calibration runs and aggregated for every problem configuration, show that the setting $\theta = 0.01$ is the best one and, most notably, $\theta = 0$ is the worst parameter value. The parametric Quade+Finner test confirms the result and, most remarkably, $\theta = 0.01$ is significantly better than the second best setting, i.e., $\theta = 0.02$, with a Finner p-value of 0.008. Moreover, since when $\theta = 0$, the θ-selection replicates the classical DE selection, we can conclude that the proposed selection scheme outperforms the classical one.

In order to furtherly understand the impact of θ on the evolution, the progress of the population diversity has been investigated by considering the evolution of the coefficient of variation of the population fitnesses. The analysis shows that increasing θ allows to quickly escape from stagnation and accelerate population convergence, thus increasing the number of restarts. However, a premature convergence can be detrimental as much as a long stagnation, thus a trade-off is needed. Since the best performances have been observed for a value of θ in the middle of the range of the tried values, this suggests that the trade-off is obtained when $\theta = 0.01$.

5) Local Search Application Scheme: The different local search application schemes considered in DEP allows to regulate if and how apply a local search when a restart is triggered.

It is worth to note that both B_LS and L_LS apply a pure

local search just after the population has been converged. Therefore, in the hypothesis that DEP evolution works well, it is plausible that the local search does not perform too many iterations and that the performances of the different schemes should be comparable.

Interestingly, the distributions of the observed performances of the three schemes on the calibration runs are almost equal. This is confirmed by the non-parametric Quade test that definitely accepts the null hypothesis (equal performances among the three settings) with a p-value of 0.66. Finally, as a further evidence, it worths to note that the first six overall ARPDs obtained in the calibration phase have $N = 100$, $\theta = 0.01$ and every other combination of local search and initialization schemes. Therefore, the strength of DEP, differently from the other evolutionary algorithms for PFSP-TFT, does not reside in a refined application of a local search procedure, but it is mainly due to its evolution scheme.

E. Computational Times

For every algorithm considered in our experimental session, Table VI shows the average CPU time (in seconds) over 10 executions spent to perform the number of allowed fitness evaluations on the first instance of every $n \times m$ problem configuration. All the executions were performed on a machine equipped with Intel Core i7-970 (6 cores and 3.2 GHz), 16 GBytes of main memory, and Linux Mint 16.

TABLE VI
COMPUTATIONAL TIMES IN SECONDS

$n \times m$	AGA	VNS_4	GM-EDA	HGM-EDA	ILS	IGA	DEP
20×5	57	43	526	88	39	54	132
20×10	117	76	693	215	112	141	229
20×20	236	186	856	557	210	190	354
50×5	149	102	1 444	237	88	108	608
50×10	263	236	2 398	597	209	238	1 016
50×20	563	484	2 856	1 577	440	474	1 345
100×5	307	230	5 279	1 038	200	2 19	1 563
100×10	535	446	8 083	2 506	392	450	2 467
100×20	1 132	950	7 518	3 403	876	907	3 133
200×10	830	919	18 788	11 675	822	884	4 180
200×20	1 718	1 865	20 205	14 131	1 829	1 856	13 727
250×10	1 206	981	29 642	17 561	939	1 147	4 803
250×20	2 296	2 461	38 738	22 894	2 314	2 171	17 872
300×10	1 654	1 358	54 476	26 374	1 236	1 305	6 032
300×20	3 160	2 752	52 105	30 768	2 558	2 719	28 453
350×10	1 627	1 673	77 564	36 192	1 636	1 390	11 405
350×20	4 441	2 571	79 979	44 755	3 228	2 955	37 952
400×10	1 774	1 791	115 135	44 553	1 800	1 640	7 501
400×20	4 103	3 889	124 942	62 954	3 524	3 583	50 858
450×10	1 931	2 063	139 616	66 868	1 768	2 052	15 136
450×20	4 616	2 468	139 472	70 026	3 804	3 914	60 222
500×20	5 099	4 377	195 076	106 429	3 910	3 853	70 506
AVG	1 719	1 451	50 700	25 700	1 451	1 466	15 432

Table VI shows that the computational times of the local search methods (VNS_4, ILS and IGA) are almost equal and comparable to the computational times of AGA, although the latter is a little slower. On the other hand, the evolutionary algorithms (GM-EDA, HGM-EDA and DEP) are, as expected, slower on average. This is mainly due to the fact that evolutionary algorithms do not allow to employ the acceleration technique for the TFT computation [33] heavily adopted by the methods based on local search schemes. However, it is worth to note that DEP is faster than both GM-EDA and HGM-EDA, especially on the larger instances.

VII. CONCLUSION AND FUTURE WORK

In this work, we have defined a new discrete meta-heuristic approach – the Differential Evolution for Permutation spaces (DEP) – and we have applied it to the permutation flowshop scheduling problem with the total flowtime criterion (PFSP-TFT).

The main contribution is the algebraic design of the differential mutation operator which allows to extend the "contour matching" property of numerical DE to every combinatorial search space representable by a finitely generated group. In order to implement the abstract algebraic differential mutation for the permutations space, a new $O(n^2)$ randomized bubble sort algorithm has been proposed. Moreover, a novel selection scheme that allows to control the evolution convergence speed and stagnation has been also introduced.

We have performed a broad experimental analysis both to tune the DEP components and to understand its dynamics. DEP has also been experimentally compared with recent state-of-the-art PFSP-TFT algorithms on a widely accepted benchmark suite. Remarkably, DEP has obtained some new best known solutions. Moreover, it is the best scheme on average on the problem configurations from 20×5 to 250×10, and it is still competitive on the larger problems with 10 machines.

Future lines of research will include: (i) the implementation of the algebraic differential mutation operator using other generating sets for the permutations space and also for other combinatorial spaces, (ii) the extension of the algebraic design to other genetic operators like crossover, classical mutation or the particle swarm update rule, and (iii) the application of DEP to other PFSP optimization criteria (the makespan objective is analyzed in [34]) or to other permutation-based problems like TSP (Traveling Salesman Problem), QAP (Quadratic Assignment Problem) and LOP (Linear Ordering Problem, preliminary results for LOP are presented in [35], [36]).

ACKNOWLEDGMENT

This work was partially supported by Italian Ministry of Education, University and Research (MIUR) under the PRIN 2010-11 grant no. 2010FP79LR_003 "Logical methods of information management", by the University of Perugia, DMI Project "Mobile Knowledge Agents in Evolutionary Environments", and by the services provided by the European Grid Infrastructure (EGI), the Italian Grid Infrastructure (IGI) and the National Grid Initiatives for the Virtual Organization (VO) COMPCHEM.

REFERENCES

[1] J. Gupta and J. E. Stafford, "Flowshop scheduling research after five decades," *European Journal of Operational Research*, vol. 169, no. 3, pp. 699–711, 2006.

[2] Q. K. Pan and R. Ruiz, "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime," *Computers & Operations Research*, vol. 40, no. 1, pp. 117–128, 2013.

[3] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano, "A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 286–300, 2014.

[4] R. M. Storn and K. V. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[5] T. Zheng and M. Yamashiro, "Solving flow shop scheduling problems by quantum differential evolutionary algorithm," *International Journal of Advanced Manufacturing Technology,*, vol. 49, no. 5, pp. 643–662, 2010.

[6] G. C. Onwubolu and D. Davendra, *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. Springer-Verlag, Berlin, 2009.

[7] V. Santucci, M. Baioletti, and A. Milani, "A differential evolution algorithm for the permutation flowshop scheduling problem with total flow time criterion," in *Proceedings of PPSN XIII, Lecture Notes in Computer Science, vol. 8672*, 2014, pp. 161–170.

[8] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer, Berlin, 2005.

[9] J. Liu and C. R. Reeves, "Constructive and composite heuristic solutions to the $p//\sum c_i$ scheduling problem," *European Journal of Operational Research*, vol. 132, no. 2, pp. 439–452, 2001.

[10] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Computers & Operations Research*, vol. 34, no. 10, pp. 3143–3153, 2007.

[11] Q. K. Pan and R. Ruiz, "Local search methods for the flowshop scheduling problem with flowtime minimization," *European Journal of Operational Research*, vol. 222, no. 1, pp. 31–43, 2012.

[12] C. Rajendran and H. Ziegler, "An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs," *European Journal of Operational Research*, vol. 103, no. 1, pp. 129–138, 1997.

[13] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[14] W. E. Costa, M. C. Goldbarg, and E. G. Goldbarg, "New vns heuristic for total flowtime flowshop scheduling problem," *Expert Systems with Applications*, vol. 39, no. 9, pp. 8149–8161, 2012.

[15] X. Xu, Z. Xu, and X. Gu, "An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization," *Expert Systems with Applications*, vol. 38, no. 7, pp. 7970–7979, 2011.

[16] S. Biswas, S. Kundu, and S. Das, "Inducing niching behavior in differential evolution through local information sharing," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 246–263, 2015.

[17] A. Milani and V. Santucci, "Community of scientist optimization: an autonomy oriented approach to distributed optimization," *AI Communications*, vol. 25, no. 2, pp. 157–172, 2012.

[18] M. Baioletti, A. Milani, V. Poggioni, and F. Rossi, "Experimental evaluation of pheromone models in acoplan," *Annals of Mathematics and Artificial Intelligence*, vol. 62, no. 3, pp. 187–217, 2011.

[19] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 154–160, 1994.

[20] Z. Čičková and S. Števo, "Flow shop scheduling using differential evolution," *Management Information Systems*, vol. 5, no. 2, pp. 8–13, 2010.

[21] X. Li and M. Yin, "An opposition-based differential evolution algorithm for permutation flowshop scheduling based on diversity measure," *Advances in Engineering Software*, vol. 55, pp. 10–31, 2013.

[22] A. Moraglio, J. Togelius, and S. Silva, "Geometric differential evolution for combinatorial and programs spaces," *Evolutionary Computation*, vol. 21, no. 4, pp. 591–624, 2013.

[23] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[24] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[25] D. Knuth, *The Art of Computer Programming, Volume 3 – Sorting and Searching*. Addison-Wesley Professional; 2nd edition, 1998.

[26] R. Sinatra, J. Gómez-Gardeñes, R. Lambiotte, V. Nicosia, and V. Latora, "Maximal-entropy random walks in complex networks with limited information," *Physical Review E*, vol. 83, no. 3, 2011.

[27] T. Murata, H. Ishibuchi, and H. Tanaka, "Genetic algorithms for flowshop scheduling problems," *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 1061–1071, 1996.

[28] A. Moraglio and R. Poli, "Geometric crossover for the permutation representation," *Intelligenza Artificiale*, vol. 5, no. 1, pp. 49–63, 2011.

[29] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[30] J. Brest, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[31] M. Birattari, "Tuning metaheuristics: A machine learning perspective,," *Studies in Computational Intelligence, vol. 197, Springer-Verlag, Berlin, Germany*, 2009.

[32] V. Santucci, M. Baioletti, and A. Milani, "Supplementary data of 'algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion'," http://www.dmi.unipg.it/baioletti/dep, 2015.

[33] X. Li, Q. Wang, and C. Wu, "Efficient composite heuristics for total flowtime minimization in permutation flow shops," *Omega*, vol. 37, no. 1, pp. 155–164, 2009.

[34] V. Santucci, M. Baioletti, and A. Milani, "Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm," *AI Communications*, forthcoming issue.

[35] M. Baioletti, A. Milani, and V. Santucci, "An algebraic differential evolution for the linear ordering problem," in *GECCO Comp '15: Proceedings of the 2015 Conference on Genetic and Evolutionary Computation Companion, ACM*, 2015, pp. 1479–1480.

[36] ——, "Linear ordering optimization with a combinatorial differential evolution," in *Proceedings of IEEE Intern. Conf. on Systems, Man, and Cybernetics (SMC)*, 2015, pp. 2135–2140.

Valentino Santucci is currently post-doctoral research fellow at Department of Mathematics and Computer Science, University of Perugia. He received a PhD in Mathematics and Computer Science from the University of Perugia in 2012. He has been visiting researcher at Hong Kong Baptist University and he is the author of more than 20 scientific publications. His main research interests involve the areas of evolutionary computation, machine learning and computational intelligence in general.

Marco Baioletti receveid a PhD in Statistics and Mathematics for the Economical and Social Research from the University of Perugia in 1993. He is currently Assistant Professor at Department of Mathematics and Computer Science, University of Perugia. His research interests include evolutionary computation, automated planning, scheduling, probabilistic logic, possibility theory. He is the author of more than 50 scientific publications on international journals and conferences.

Alfredo Milani IEEE Member, is Associate Professor at Department of Computer Science, University of Perugia where he heads the Knowledge & IT Lab. His research interests lie in the area of artificial intelligence, including planning and agent systems, evolutionary meta-heuristics, data mining and semantic proximity measures. He chaired or co-chaired several international conferences in the AI area. He has been Visiting Associate Professor at Hong Kong Baptist University and Visiting Scientist at Jet Propulsion Laboratory, Pasadena, USA.