Solving Permutation Flowshop Scheduling Problems with a Discrete Differential Evolution Algorithm

Valentino Santucci [a]
Marco Baioletti [a]
Alfredo Milani [a,b]
[a] *Dept. of Mathematics and Computer Science*
University of Perugia
Perugia (Italy)
[b] *Department of Computer Science*
Hong Kong Baptist University
Hong Kong (China)
E-mail:
{*milani,baioletti,valentino.santucci*}*@dmi.unipg.it*

In this paper a new discrete Differential Evolution algorithm for the Permutation Flowshop Scheduling Problem with the total flowtime and makespan criteria is proposed. The core of the algorithm is the distance-based differential mutation operator defined by means of a new randomized bubble sort algorithm. This mutation scheme allows the Differential Evolution to directly navigate the permutations search space. Experiments were held on a well known benchmarks suite and they show that the proposal reaches very good performances compared to other state-of-the-art algorithms. The results are particularly satisfactory on the total flowtime criterion where also new known optima have been found.

Keywords: Permutation Flowshop Scheduling Problem, Differential Evolution, Permutation-based Optimization

1. Introduction and Related Works

The Permutation Flowshop Scheduling Problem (PFSP) is a type of scheduling problem widely encountered in areas such as manufacturing and large scale products fabrication [1]. In the PFSP there are n jobs $J_1, \ldots, J_n$, each of them composed by a sequence of m operations. O_{ij} is the i-th operation of job J_j and p_{ij} is its processing time. There are m machines $M_1, \ldots, M_m$ and a generic operation O_{ij} can be executed only by machine M_i. Moreover, the execution of any operation cannot be interrupted (no pre-emption) and job passing is not allowed, i.e., the jobs have to be executed following the same order in every machine.

Hence, the goal of PFSP is to determine the best permutation $\pi = \langle \pi(1), \ldots, \pi(n) \rangle$ of the n jobs according to some objective function.

Two important criteria are to minimize the total flowtime (TFT)

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

and the makespan

$$f_{max}(\pi) = \max_{j=1,\ldots,n} c(m, \pi(j)) = c(m, \pi(n)) \tag{2}$$

where, in both equations, $c(i, \pi(j))$ is the completion time of job $\pi(j)$ on machine i and is recursively calculated as

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

in the general case when $i, j > 1$; while in terminal cases:

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

Finally, the last equality in equation (2) follows from the fact that $p_{ij} \geq 0$ for $1 \leq i \leq m$ and $1 \leq j \leq n$.

The PFSP with the TFT criterion has been shown to be NP hard for two or more machines [2], while the makespan criterion is slightly easier,

AI Communications
ISSN 0921-7126, IOS Press.

because can be solved in polynomial time for two machines, but remains NP hard for $m > 2$. Therefore, even due to their practical interest, many researches have been devoted to finding high quality and near optimal solutions by means of heuristic or meta-heuristic approaches [3,5,4].

A large number of methods for the PFSP-Makespan have been described and compared in [4] where it is shown that the heuristic NEH [7] is very effective and ILS was the best meta-heuristic algorithm [27]. More recently an iterated greedy (IG) algorithm has been introduced in [6] and it can be considered as the state-of-the-art for the makespan objective. IG adopts a simulated annealing like acceptance criterion and iteratively alternates a destruction-construction procedure and a local search scheme. The initial solution is built with the NEH heuristic.

A report of the state-of-the-art methods for PFSP-TFT has been recently provided in [5] where it is shown that the most performing meta-heuristics are: VNS_4 [10], AGA [11] and GM-EDA together with its hybrid variant HGM-EDA [5]. VNS_4 applies a variable neighborhood search (VNS) to an initial permutation built by means of a constructive heuristic called LR(n/m) [12]. AGA is an asynchronous genetic algorithm hybridized with VNS. GM-EDA is an estimation of distribution algorithm that adopts a probabilistic model for the permutations space known as generalized Mallows model, while HGM-EDA represents the hybridization of GM-EDA with a VNS scheme.

Differential Evolution [13] is a popular evolutionary algorithm (see for example [14,15]) for continuous optimization problems. Although its effectiveness in numerical spaces, DE applications to combinatorial problems, and in particular to permutation-based problems, are still unsatisfactory. To the best of our knowledge, all the DE algorithms for the PFSP proposed in literature (see for example the schemes reported in [16] or the more recents [17,18]) adopt some transformation scheme to encode permutations into numerical vectors. This distinction between the phenotypic and the genotypic space [20] introduces large plateaus in the numerical landscape and is probably the reason of their poor performances. To address this issue, in this paper we propose a discrete DE scheme for the PFSP problem, using both optimization criteria, that works directly on the permutations space. Since the differential mutation operator has been generally considered the key component of DE [19], our approach mainly relies on a differential mutation operator that directly handles permutations, thus trying to fruitfully bring the DE search properties from the numerical space to the combinatorial space of permutations. Furthermore, a new $O(n^2)$ randomized bubble sort algorithm is provided.

The rest of the paper is organized as follows. A short overview of Differential Evolution and a short introduction to group theory and to the permutation group are provided in Sections 2 and 3. The permutation-based differential mutation operator and the new randomized bubble sort algorithm are introduced and motivated in Section 4. The full DE scheme for PFSP is described in Section 5. An experimental analysis of the proposed approach for both criteria is provided in Section 6. Finally, conclusions are drawn in Section 7 where some future lines of research are also depicted.

2. The Differential Evolution algorithm

In this section we provide a short introduction to original and numerical Differential Evolution (DE) algorithm (for more details see [19]). DE is a simple and powerful population based meta-heuristic for optimizing non-linear and even non-differentiable real functions. The core component of DE resides in its differential mutation operator that allows DE to probe the search space by automatically tuning the step size and the orientation basing on the fitness landscape at hand.

DE initially generates a random population of NP candidate solutions $x_1, \ldots, x_{NP} \in \mathbb{R}^n$ uniformly distributed in the solutions space. At each generation, DE performs the mutation and the crossover operations to produce a *trial* vector u_i for each individual x_i, called *target* vector, in the current population. Each target vector is then replaced in the next generation by the associated trial vector if and only if the produced trial presents a better fitness than the target. This process is iteratively repeated, through the so called generations, until a stop criterion is met (e.g., a given amount of fitness evaluations has been performed).

The mutation phase generates a *mutant* vector v_i for each target individual x_i. The simplest and most used mutation scheme is "rand/1" and com-

putes the mutant as follows:

$$v_i = x_{r_0} + F \cdot (x_{r_1} - x_{r_2}) \tag{3}$$

where r_0, r_1, r_2 are three random integers in $[1, NP]$ mutually different among them and with respect to i. x_{r_0} is called *base vector*, while $x_{r_1} - x_{r_2}$ is the *difference vector*, and $F > 0$ is the *scale factor* parameter.

This differential mutation is the key operator of DE. Indeed, it confers to the algorithm the ability to automatically adapt the mutation step size and orientation to the fitness landscape (see [19]). As a consequence, the effect of this operator is that the DE search automatically shades from an explorative to an exploitative behaviour with the passing of generations.

After the mutation, a crossover operator generates a population of NP trial vectors, i.e. u_i, by recombining each pair composed by the generated mutant v_i and its corresponding target x_i. Indeed, each trial vector is formed by taking some components from the target vector and some other ones from the mutant, according to the crossover probability $CR \in [0, 1]$.

The most used crossover operator is the binomial one and produces a trial vector u_i whose j–th component is:

$$u_{i,j} = \begin{cases} v_{i,j} \text{ if } \theta_{1,j} \leq CR \text{ or } \theta_{2,j} = j \\ x_{i,j} \text{ otherwise} \end{cases} \tag{4}$$

where $\theta_{1,j} \in [0, 1]$ is a random number generated for each dimension j and $\theta_{2,j} \in [1, D]$ is a random integer generated for each trial vector which ensures that u_i inherits at least one component from the mutant v_i.

Finally, in the selection phase, the new population individual x'_i is the best between x_i and u_i, $i = 1, \ldots, NP$. More formally, in the case of a minimization problem with fitness function f, x'_i is computed as follows

$$x'_i = \begin{cases} u_i \text{ if } f(u_i) \leq f(x_i) \\ x_i \text{ otherwise} \end{cases} \tag{5}$$

3. A short introduction to group theory and permutations

In this section we provide a short introduction to group theory and, in particular, to the permutation group.

A group G is a set provided with an internal operation $\star : G \times G \to G$, such that

1. $\star$ is associative
2. there exists an element $e \in G$ which is the neutral element for $\star$, i.e., $x \star e = e \star x = x$ for all $x \in G$
3. for all $x \in G$, there exists the inverse element x^{-1} of x in G, i.e., $x \star x^{-1} = x^{-1} \star x = e$

If the operation $\star$ is commutative, G is said Abelian.

A set $S \subseteq G$, such that for all $s \in S$, also $s^{-1} \in S$, is called a generator set for G if for every element $x \in G$ there exists a finite sequence of elements $s_1, s_2, \ldots, s_k \in S$ such that $x = s_1 \star s_2 \star \cdots \star s_k$. Hence, S is a generator set for G if every element $x \in G$ can be decomposed as a product of a finite number of elements of S.

This decomposition in general is not unique. Moreover, two decompositions for the same element x have not necessarily the same length. In general, the problem of finding a decomposition can be a hard computational problem.

Anyway, given a generator set S for a group G, it is possible to define the Cayley graph Γ, i.e., a regular graph whose vertices are the elements of G and, for any $x \in G$ and $s \in S$, there is an edge labeled with s which connects x and $x \star s$. In this graph, it is possible to define a distance function $d(x, y)$ for each $x, y \in G$ as the length of a shortest path from x to y. Note that a path from e to x corresponds to a decomposition of x, indeed the sequence of labels $s_1, \ldots, s_L$ is such that $x = e \star s_1 \star \cdots \star s_L$. Hence, the length of a shortest decomposition of $x \in G$ equals to $d(e, x)$. The maximum distance between two elements $x, y \in G$ is called diameter of Γ.

Now we introduce the group of permutations $S(n)$ for the set $[n] = \{1, 2, \ldots, n\}$. A permutation is a bijective function $\pi : [n] \to [n]$ and it can be interpreted as an ordering for the set $[n]$, i.e. $\pi(x) = y$ means that the element y is at x-th place. The group $S(n)$ is then composed by all the permutations and its operation is the ordinary composition operator $\circ$ between functions, i.e. given $\pi_1, \pi_2 \in S(n)$ $(\pi_1 \circ \pi_2)(x) = \pi_1(\pi_2(x))$ for all $x \in [n]$. The neutral element is the identity permutation e, which is defined as $e(x) = x$, for all $x \in [n]$. The inverse permutation π^{-1} of a permutation π is just the ordinary functional inverse of π, i.e., $\pi^{-1}(y) = x$ if and only $\pi(x) = y$, for all $x, y \in [n]$.

Note that the cardinality of $S(n)$ is $n!$ and that $S(n)$ is not Abelian, when $n \geq 3$.

There exists many generator sets for $S(n)$. Among them, the most important are transpositions, simple transpositions and insertions.

Given $i, j \in X$, with $i < j$, a transposition is a permutation $\tau^{i,j}$ obtained from e by swapping the elements at places i and j. The set of all the transpositions $T = \{\tau^{i,j} : 1 \leq i < j \leq n\}$ generates $S(n)$ because it is known that every permutation can be written as a product of disjoint cycles and each cycle can be written as product of transpositions [30]. Moreover, it is easy to prove that the diameter of the corresponding Cayley graph is $n - 1$. The distance induced is called Cayley distance and it can be computed with a selection-sort like algorithm [22]. The cardinality of T is $\binom{n}{2}$.
A transposition is simple if $j = i + 1$, i.e. it swaps two consecutive elements. Since any transposition can be written as product of a finite number of simple transpositions, also the set $ST = \{\tau^{i,i+1} : 1 \leq i \leq n - 1\}$ generates $S(n)$. The cardinality of ST is $n - 1$, while the diameter of the induced Cayley graph is $\binom{n}{2}$. The induced distance between two permutations π_1 and π_2 is called Kendall-τ distance and denoted by d_K and it can be computed with a bubble-sort like algorithm [22]. An important property of d_K is that $d_K(e, \pi)$ is equal to the number of inversions of π, i.e. the number of pairs (i, j) such that $i < j$ and $\pi(i) > \pi(j)$.
Finally, given $i, j \in X$, with $i \neq j$, the insertion $\psi^{i,j}$ is a permutation obtained from e by shifting the element at place j to place i. The set $I = \{\psi^{i,j} : i \neq j \text{ and } 1 \leq i, j \leq n\}$ contains ST, because each simple transposition $\tau^{i,i+1}$ is the particular insertion $\psi^{i,i+1}$, hence I generates $S(n)$. Its cardinality is $(n - 1)^2$ and, again, the diameter of its Cayley graph is $n - 1$. The induced distance is called Ulam distance and it can be computed with a insertion-sort like algorithm [22].

4. Differential Mutation in the Permutations Space

The purpose of this paper is to define a differential evolution algorithm to optimize permutation functions, like those used in permutation based scheduling. However, it is preferable to avoid to use a numerical encoding of permutations, as done for instance in [16,17,18], because in this technique a permutation can be encoded as infinitely many numerical vectors and hence it induces large plateau in the search spaces. Our proposal is instead of directly working on the permutations space.

The key component is a differential mutation scheme defined on the permutations, analogous to (3).

The sum and the difference of two permutations can be easily defined: the sum $\pi_1 \oplus \pi_2 \in S(n)$ is defined as $\pi_1 \circ \pi_2$, while the difference $\pi_1 \ominus \pi_2$ can be correspondingly defined as $\pi_2^{-1} \circ \pi_1$. Indeed

$$\pi_1 \oplus (\pi_2 \ominus \pi_1) = \pi_1 \circ \left(\pi_1^{-1} \circ \pi_2\right) = \pi_2$$

analogously in the numerical case.

The differential mutation operator requires also to multiply the difference of two permutations, i.e. a third permutation π, for a scale factor $F \in [0, 1]$. Algebraically, to the best of our knowledge, this operation has never been studied. We provide an operative definition of this operation which is based on the choice of a generator set S for the permutation group. Indeed $F \cdot \pi$ can be computed in the following way

1. decompose π as $s_1 \circ s_2 \circ \cdots \circ s_L$, where $s_1, \ldots, s_L \in S$ and L is as short as possible (i.e. we take a shortest length decomposition of π);
2. compute $k = \lceil F \cdot L \rceil$;
3. return the permutation $s_1 \circ s_2 \circ \cdots \circ s_k$.

Since the decomposition is not unique, the result also depends on the choice of the method to obtain a decomposition of π.

Hence the differential mutation scheme is

$$\nu_i = \pi_{r_0} \circ \left(F \cdot \left(\pi_{r_2}^{-1} \circ \pi_{r_1}\right)\right) \tag{6}$$

It is worth to notice that the difference between two permutations π_1 and π_2 can be represented as any of the shortest paths from π_2 to π_1 in the Cayley graph. The sequences of labels $s_1, \ldots, s_L$ of such paths are different, but for all of them the product of labels $s_1 \circ s_2 \circ \cdots \circ s_L$ is equal to $\pi_2^{-1} \circ \pi_1$. Moreover $F \cdot (\pi_2^{-1} \circ \pi_1)$ corresponds to select one of those paths and truncate it after $k = \lceil F \cdot L \rceil$ edges.

We have decided to use ST as generator set due to the observation that a smaller number of generators induces a higher search space diameter and generally longer paths connecting two so-

lutions. This looks to be more suited for a differential mutation operator. Indeed, a shortest path in the ST space is generally longer than in the two other spaces and is composed by "many little steps", each of them more likely to produce slight variations in the objective function. The truncation operator, if performed on spaces with smaller diameter and shorter paths, like T and I, would have a reduced number of possible values and we expect that it would produce worst results. Note also that ST induces a smaller "branching factor" on the search space, thus reducing the arbitrariness of the navigation. These considerations can be in general extended to other combinatorial optimization problems.

A shortest decomposition of a permutation π in terms of simple transposition can be computed using the well known bubble sort algorithm or some of its deterministic variants (cocktail sort, gnome sort, etc.). However, since DE is a stochastic algorithm, it is better to use a randomly generated shortest decomposition. In fact, since the purpose of decomposing π is to truncate it up to the k–th position, deterministic algorithms will tend to put in the first positions of the decomposition almost always the same simple transpositions, while a randomized decomposition algorithm produces decompositions in a more fair way. Hence, we propose a randomized version of bubble sort that is outlined in Algorithm 1.

Algorithm 1 Randomized Bubble Sort

```
1: procedure RandBS(π)
2:   CC ← ⟨ ⟩
3:   INV ← {i : π(i) > π(i + 1)}
4:   while INV ≠ ∅ do
5:     i ← RemoveRandomElement(INV)
6:     Swap π(i) and π(i + 1)
7:     Append τ^{i,i+1} to CC
8:     if i > 0 and i − 1 ∉ LST and π(i − 1) > π(i) then
9:       Add i − 1 to INV
10:    end if
11:    if i < n − 1 and i + 1 ∉ LST and π(i + 1) > π(i + 2) then
12:      Add i + 1 to INV
13:    end if
14:  end while
15:  return Reverse(CC)
16: end procedure
```

The *RandBS* algorithm sorts the permutation π obtaining the "sorted" permutation e using the optimal number $L = d_K(\pi, e)$ of adjacent swaps. Indeed, INV initially contains all the adjacent inversions of π, i.e. the positions i such that $\pi(i) > \pi(i+1)$. Then, at each iteration of the while loop:

1. an index i is randomly drawn from INV and the simple transposition $\tau^{i,i+1}$ is applied to π. The effect is that the Kendall-τ distance from π to e is reduced by 1.
2. $\tau^{i,i+1}$ is added to CC
3. INV is updated, because at most two new consecutive inversions can appear in π

The final result of *RandBS* is the reverse of CC. In fact, CC contains the sequences of simple transpositions $s_1, \ldots, s_L$ such that $\pi \circ s_1 \circ s_2 \circ \cdots \circ s_L = e$, hence a decomposition of π is obtained as

$$\pi = s_L^{-1} \circ s_{L-1}^{-1} \circ \cdots \circ s_1^{-1}$$

Anyway, each simple transposition is the inverse of itself, therefore

$$\pi = s_L \circ s_{L-1} \circ \cdots \circ s_1$$

i.e., the reverse of the list CC.

Since the number of iterations is $L \leq \binom{n}{2} = O(n^2)$, which corresponds to the diameter of ST, and all the operations inside the loop can be implemented in $O(1)$, the time complexity of *RandBS* is $O(n^2)$, as the one of its classical counterpart.

Furthermore, it is worthwhile to notice that the elements of CC corresponds to a "never go back" random walk from e towards π in the subgraph of Γ induced by the "interval"

$$[e, \pi]_K = \{\sigma \in S(n) : d_K(e, \sigma) + d_K(\sigma, \pi) = d_K(e, \pi)\}. \tag{7}$$

Finally, the differential mutation can be computed by the Algorithm 2.

Algorithm 2 Differential Mutation

```
1: procedure DiffMut(i, F)
2:   Find r0, r1, r2 different to i and to each other
3:   δ ← π_{r2}^{-1} ∘ π_{r1}
4:   S ← RandBS(δ)
5:   L ← Length(S)
6:   k ← ⌈F · L⌉
7:   ν ← π_{r0}
8:   for i ← 1 to k do
9:     Apply S_i to ν
10:  end for
11:  return ν
12: end procedure
```

For the sake of clarity we provide an illustrative example of how the differential mutation works. Suppose that $n = 5$, $F = 0.5$, and the given permutations are $\pi_0 = \langle 3, 4, 1, 2, 5 \rangle$, $\pi_1 = \langle 1, 4, 2, 5, 3 \rangle$ 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 compute $\pi_1 \ominus \pi_2$, that is, $\pi_2^{-1} \circ \pi_1 = \langle 3, 4, 5, 1, 2 \rangle$. Now, a possible result of $RandBS(\pi_2^{-1} \circ \pi_1)$ is the decomposition $\langle \tau^{2,3}, \tau^{3,4}, \tau^{1,2}, \tau^{2,3}, \tau^{4,5}, \tau^{3,4} \rangle$ which has length 6. Therefore, the truncation $0.5 \cdot (\pi_1 \ominus \pi_2)$ is $\langle \tau^{2,3}, \tau^{3,4}, \tau^{1,2} \rangle$ that evaluates to $\tau^{2,3} \circ \tau^{3,4} \circ \tau^{1,2} = \langle 3, 1, 4, 2, 5 \rangle$. Finally, by composing π_0 with the truncated difference, we obtain $\nu = \langle 1, 3, 2, 4, 5 \rangle$

5. Differential Evolution for Permutations

The Differential Evolution for the Permutations space (DEP), outlined in Algorithm 3, directly evolves a population of NP permutations $\pi_1, \ldots, \pi_{NP}$. Its main scheme resembles that of the classical DE with the introduction of a restart mechanism and a memetic local search procedure. Moreover, important variations have been made to the population initialization and to the genetic operators of mutation, crossover and selection. All these components are described in the following subsections. Note that, both in Algorithm 3 and in the following descriptions, depending on the chosen objective function, f can interchangeably refer to f_{sum} or f_{max}.

Algorithm 3 DE for Permutations

```
Initialize Population
while evaluations budget is not exhausted do
  for i ← 1 to NP do
    ν_i ← DifferentialMutation(i)
    υ_i^(1), υ_i^(2) ← Crossover(π_i, ν_i)
    Evaluate f(υ_i^(1)) and f(υ_i^(2))
  end for
  for i ← 1 to NP do
    π_i ← Selection(π_i, υ_i^(1), υ_i^(2))
  end for
  if restart criterion then
    Perform a Local Search on π_best
    Restart Population
  end if
end while
```

5.1. Initialization

The population is initialized with $NP - 1$ random permutations (obtained by means of the uniformly random permutation generator known as Fisher-Yates shuffle [23]) and the remaining permutation is built using the appropriate constructive heuristic for the PFSP objective at hand, i.e., $LR(n/m)$ [12] for the total flowtime and NEH [7] for the makespan.

5.2. Differential Mutation

For each population individual π_i, a mutant permutation ν_i is generated according to Algorithm 2.

Furthermore, in order to avoid the setting of the scale factor F, the popular self-adaptive scheme proposed in jDE [24] has been used for its adaptation during the evolution. Basically, it introduces a self-adapting parameter F_i for each individual. Just before the mutation, a temporary F_{mutant} is generated according to

$$F_{mutant} \leftarrow \begin{cases} 0.1 + r_1 \cdot 0.9 & \text{if } r_2 < 0.1 \\ F_i & \text{otherwise} \end{cases}$$

where r_1, r_2 are two random numbers in $[0, 1]$. The mutation is performed by using F_{mutant}, and, in the case that one of the offsprings replaces the original population individual, also F_{mutant} replaces F_i.

5.3. 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) proposed in [25] and used by AGA [11]. Differently from the classical DE crossover, TPII produces two trial individuals, i.e., $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$.

This operator works as follows:

- two indices j, k, such that $1 < j < k < n$, are randomly generated
- for $h \in \{j, \ldots, k\}$, $\upsilon_i^{(1)}(h) = \pi_i(h)$
- all the other jobs, i.e. the jobs $\pi_i(h)$ for $h \notin \{j, \ldots, k\}$, are placed in the free places of $\upsilon_i^{(1)}$ using the relative order of their appearance in ν_i, i.e., the job $\pi_i(h)$, for $h \notin \{j, \ldots, k\}$, with the lowest index in ν_i goes in the leftmost free position of $\upsilon_i^{(1)}$, and so on;
- $\upsilon_i^{(2)}$ is computed in the same way but by reversing the role of π_i and ν_i

For instance, if $\pi_i = \langle 3, 4 | 1, 2, 8 | 7, 6, 5 \rangle$, $\nu_i = \langle 8, 2 | 3, 6, 5 | 4, 1, 7 \rangle$, $j = 3, k = 5$, then $\upsilon_i^{(1)} = \langle 3, 6 | 1, 2, 8 | 5, 4, 7 \rangle$ and $\upsilon_i^{(2)} = \langle 4, 1 | 3, 6, 5 | 2, 8, 7 \rangle$.

5.4. Selection

In order to choose the trial v_i that will compete with π_i, a preliminary selection between the two trial individuals is performed according to:

$$v_i = \text{argmin}\left\{f\left(v_i^{(1)}\right), f\left(v_i^{(2)}\right)\right\} \tag{8}$$

Then, the new population individual π_i' is chosen by a "biased" selection between v_i and π_i performed according to:

$$\pi_i' = \begin{cases} v_i & \text{if } f(v_i) < f(\pi_i) \text{ or } r < \max\{0, \alpha - \Delta_i\} \\ \pi_i & \text{otherwise} \end{cases} \tag{9}$$

where r is a random number in $[0, 1]$, Δ_i is the relative fitness variation $\dfrac{f(v_i) - f(\pi_i)}{f(\pi_i)}$, and $\alpha \in [0, 1]$ is a selection parameter.

Similarly to classical DE selection, v_i enters the next generation population if it is fitter than π_i. Otherwise, v_i may be selected with a small probability that linearly shades from α when $\Delta_i = 0$ to 0 when $\Delta_i = \alpha$. It is worthwhile to note that, when $\alpha = 0$, the classical DE selection scheme is reproduced.

This criterion allows to mitigate the stagnation of the population observed in some preliminary experiments which used the classical selection.

5.5. Restart

A restart mechanism has been introduced in order to completely avoid the stagnation of the population. When all the population elements are equal, the algorithm is not able to evolve anymore. In fact, the differential mutation applied to three identical individuals always produce the same individual. Also the crossover has the same property. Hence, in all the generations, the population remains unchanged[1]. In this situation, a restart mechanism is indispensable.

Instead of checking that all the individual are equal, we check if all the fitness values are equal. Experimentally, the two conditions are equivalent in almost all the cases, but the latter is more efficient to check. Hence, when the restart condition is verified, one of the individual is kept and the other $NP - 1$ permutations are randomly reinitialized.

[1]This situation happens also in the classical DE

5.6. Local Search

A local search procedure is performed at every restart and it is applied to the individual kept in the restart phase.

The local search scheme employed is similar to VNS_4 [10] without shakes. A greedy local search using the interchange neighborhood is carried out until a local minimum is found. Then, the best neighbor in its insertion neighborhood is chosen and the process is iterated until a local minimum for both neighborhoods is reached.

There are two possible local search application strategies: Baldwinian (B_LS) and Lamarckian (L_LS) strategies. In the first strategy, the result of the local search does not enter in the population and is used only to update the global best of DEP. In this way, the evolution is not affected by the local search, even if it consumes computational resources.

In the second strategy, the improved individual enters the population, so it can also modify the dynamics of the evolution.

6. Experiments

The performances of the proposed DEP algorithm for PFSP have been evaluated and investigated on both the total flowtime (TFT) and the makespan criteria.

The well known 120 PFSP instances proposed by Taillard [26] have been employed as benchmark suite. This includes different instances with $n \in \{20, 50, 100, 200, 500\}$, $m \in \{5, 10, 20\}$ and where the processing times p_{ij}, with $1 \leq i \leq m$ and $1 \leq j \leq n$, have been randomly sampled from the integer interval $[1, 99]$. The wide adoption of this benchmark suite in the PFSP literature (see for example [5] and [6]) also allows a fair comparison with the state-of-the-art algorithms for both the objectives.

It worths to note that the effectiveness of an algorithm on PFSP-TFT does not necessarily imply its effectiveness on PFSP-Makespan and vice versa. Indeed, PFSP-TFT and PFSP-Makespan are often studied independently to each other (see for example [5] for TFT and [6] for makespan). A further evidence of the uncorrelation between the two objectives can be found in the extensive scientific production for the multi-objective

PFSP (see for example [31,32,33,34]). Also the fitness landscapes of the two problems are likely to have different characteristics. Indeed, given the same problem instance, both PFSP-TFT and PFSP-Makespan have the same number of feasible solutions (i.e., $n!$), but the makespan is just $c(m, \pi(n))$, and the number of its possible values is much smaller than the total flow time, which is $\sum_{j=1}^{n} c(m, \pi(j))$. Hence, PFSP-TFT is likely to have a larger number of the allowed fitness values than PFSP-Makespan.

This in turn impacts on the quantities of equal quality solutions, thus it is more likely to have more and larger plateaus on PFSP-Makespan and, consequently, a more rugged landscape on PFSP-TFT.

Therefore, DEP has been separately calibrated on the two optimization criteria and has been compared with the different state-of-the-art algorithms for both PFSP-TFT and PFSP-Makespan.

DEP parameters calibration has been performed on an additional set of PFSP instances randomly generated using the same generator scheme of Taillard [26]. According to [35], this set of tuning instances is representative of the benchmark suite used for testing comparison and allows to avoid the "over-tuning" phenomenon.

Furthermore, due to the stochastic nature of the meta-heuristics considered, both the calibration and the testing experiments have been performed using multiple executions of them. A maximum number of fitness evaluations has been used as a termination criterion for every considered algorithm and has been set according to [5]. In order to be self-contained the budget of fitness evaluations for each $n \times m$ problem configuration is reported in Table 1.

Table 1
Budgets of Fitness Evaluations

$n \times m$	Evaluations Cap	$n \times m$	Evaluations Cap
20×5	182 224 100	100×5	235 879 800
20×10	224 784 800	100×10	266 211 000
20×20	256 896 400	100×20	283 040 000
50×5	220 712 150	200×10	272 515 500
50×10	256 208 100	200×20	287 728 850
50×20	275 954 150	500×20	260 316 750

The performance measure employed is the commonly adopted average relative percentage deviation (ARPD):

$$ARPD = \frac{1}{k} \sum_{i=1}^{k} \frac{(Alg_i - Best) \times 100}{Best} \qquad (10)$$

where Alg_i is the best objective value found by the algorithm Alg in its i-th run, and $Best$ is a reference objective value for the problem instance at hand.

Finally, in order to detect significant differences, the ARPDs obtained by the different competitor algorithms have been statistically compared using non-parametric statistical tests as suggested in [28].

The rest of this section is organized as follows. Subsection 6.1 describes DEP calibration for both the optimization criteria. Subsection 6.2 and 6.3 provides the experimental results respectively for PFSP-TFT and PFSP-Makespan. A further experimental analysis on the impact of the newly proposed discrete differential mutation is provided in subsection 6.4, while the computational times of DEP are investigated in subsection 6.5.

6.1. DEP Parameters Calibration

DEP has four parameters: the scale factor F, the population size NP, the selection parameter α, and the local search application strategy B_LS or L_LS.

While, as described in Section 5.2, the popular self-adaptive scheme proposed in jDE [24] has been adopted in order to automatically tune F during the evolution, the other three parameters have been calibrated by means of a full factorial experimental analysis. The considered values for each one of the involved parameters have been selected after some preliminary experiments and they are:

- for the population size NP: 20, 50, 100, and 200;
- for the selection parameter α: 0, 0.01, and 0.02;
- for the local search application strategy: B_LS and L_LS.

Therefore, there are $4 \times 3 \times 2 = 24$ DEP settings to experimentally compare.

As previously described, in order to avoid the "over-tuning" phenomenon, the calibration has been performed on different PFSP instances with respect to the ones adopted in the testing experiments of Sections 6.2 and 6.3. Indeed, 10 new instances have been generated for 11 different prob-

lem configurations: all the $n \times m$ configurations with $n \in \{20, 50, 100, 200\}$ and $m \in \{5, 10, 20\}$ except for the configuration 200×5 (note that these are the same problem configurations considered in the Taillard benchmark suite [26] except 500×20). The calibration instances have been randomly generated using the same probability distribution of Taillard in order to have a representative, though different, set with respect to the testing suite.

As motivated before, two different calibrations have been performed separately for the two optimization criteria TFT and makespan. Moreover, the experimental design has been modeled with the aim of providing two DEP settings, one for PFSP-TFT and one for PFSP-Makespan, that (hopefully) perform well across different problem configurations.

For both the objectives, the performances have been evaluated as follows. Each DEP setting has been run once per problem instance and an ARPD value for each DEP setting and each $n \times m$ problem configuration has been obtained by averaging the relative percentage deviations obtained on the ten distinct problem instances. According to [35, Th. 4.12], given a budget of executions, this experimental setting is the one that minimize the variance of the averaged performance estimator ARPD. Therefore, the calibration experimental session consisted of a total of 24 (DEP settings) $\times$ 10 (instances) $\times$ 11 (configurations) $\times$ 2 (objectives) = 5 280 DEP executions.

Since each one of the 24 DEP settings has 11 ARPD values obtained on problem configurations with different sizes and difficulties, according to [28], the non-parametric Quade test has been employed in order to perform a statistical comparison of the DEP settings both for PFSP-TFT and PFSP-Makespan. Indeed, the Quade test allows to weigh the ARPDs basing on the problem difficulty (actually, the weights are based on the range of performance variations registered on the different instances of the same problem configuration). Moreover, since the statistical test loses significance when the number of competitors is larger than the number of problems, only the DEP settings with the best ARPD on at least one problem configuration have been selected for comparison. The Quade test produces: (i) a weighted average rank for each selected competitor that allows to rank the DEP settings basing on their effectiveness, and (ii) a p-value indicating the probability that the performances of all the compared DEP settings are not significantly different. Lastly, in the case of significant difference among all the competitors, the best DEP setting has been compared with the others using the Finner post-hoc procedure as suggested in [28].

Tables 2 and 3 provide the experimental results of the selected DEP settings respectively for PFSP-TFT and PFSP-Makespan. Every row relates to a different DEP setting and are ordered basing on their Quade average ranks. Also the average ARPD and the Finner p-value with respect to the best DEP setting are reported as summary statistics. The ARPDs obtained on the different problem configurations are also reported. The best results are indicated in bold. Finally, the results of the 20 jobs problem configurations are not provided because all the settings perform the same on these instances.

Regarding PFSP-TFT, as shown by Table 2, the best DEP setting, both with respect to the Quade average rank and the overall ARPD, is $(NP = 100, \alpha = 0.01, B\_LS)$. Interestingly, this setting is the best one on the 50×20 and 200×20 problems and it is among the first three settings on all the other problems. Moreover, the very small Quade p-value indicates that there are significant differences among the seven parameters settings analyzed and the post-hoc Finner procedure cuts off the last three settings reported in Table 2, thus leaving only four DEP settings with not enough evidence of statistical difference among them. Interestingly, these four settings share the same α value, i.e., $\alpha = 0.01$. This shows that, in the case of PFSP-TFT, the biased selection introduced in this work allows to improve the results with respect to the classical DE selection (obtained with $\alpha = 0$).

Regarding PFSP-Makespan, Table 3 shows that the best DEP setting, both with respect to the Quade average rank and the overall ARPD, is $(NP = 20, \alpha = 0.01, L\_LS)$. This setting is also the best one on four problem configurations and it is among the first three settings on the other problems. Differently from the PFSP-TFT, the Quade p-value does not indicate enough statistical evidence of performance differences among the selected five DEP settings. However, all the five DEP settings selected show a preference towards a small population size in the case of PFSP-Makespan.

Summarizing the DEP parameters setting selected for PFSP-TFT is

Table 2
DEP Calibration Results for Total Flowtime Criterion

DEP Setting			Summary Results			Problem ARPDs							
NP	α	LS	Quade AvgRank	Overall ARPD	Finner p-value	50×5	50×10	50×20	100×5	100×10	100×20	200×10	200×20
100	**0.01**	**B_LS**	**2.00**	**0.10**	-	0.10	0.13	**0.06**	0.04	0.10	0.09	0.22	**0.08**
100	0.01	L_LS	2.78	0.13	0.45	0.12	0.20	0.16	**0.02**	0.14	0.10	0.12	0.17
200	0.01	L_LS	3.17	0.71	0.30	0.10	0.28	0.23	0.14	**0.08**	0.42	0.60	3.85
50	0.01	B_LS	3.39	0.22	0.25	0.11	0.23	**0.06**	0.09	0.27	**0.08**	0.54	0.38
200	0.02	L_LS	5.47	0.50	<0.01	0.08	0.17	0.13	0.05	0.11	0.31	**0.04**	3.10
50	0.02	B_LS	5.50	0.71	<0.01	0.11	**0.12**	0.18	0.21	0.30	0.33	0.57	3.88
20	0.01	L_LS	5.69	0.35	<0.01	**0.04**	0.28	0.18	0.23	0.45	0.34	0.58	0.67

Quade p-value: 0.001

Table 3
DEP Calibration Results for Makespan Criterion

DEP Setting			Summary Results			Problem ARPDs							
NP	α	LS	Quade AvgRank	Overall ARPD	Finner p-value	50×5	50×10	50×20	100×5	100×10	100×20	200×10	200×20
20	**0.01**	**L_LS**	**1.92**	**0.07**	-	**0.00**	**0.02**	0.27	**0.00**	0.05	0.20	0.02	**0.03**
20	0	B_LS	3.06	0.21	0.20	**0.00**	**0.02**	0.47	**0.00**	0.08	0.80	**0.00**	0.28
20	0.02	L_LS	3.14	0.19	0.22	**0.00**	0.53	**0.14**	**0.00**	0.14	0.35	0.18	0.18
50	0.01	L_LS	3.36	0.22	0.20	**0.00**	0.15	0.26	**0.00**	0.15	**0.06**	0.06	1.07
20	0	L_LS	3.53	0.24	0.27	**0.00**	0.11	0.62	**0.00**	**0.01**	0.75	0.03	0.38

Quade p-value: 0.393

$$(NP = 100, \alpha = 0.01, B\_LS),$$

while for PFSP-Makespan is

$$(NP = 20, \alpha = 0.01, L\_LS).$$

6.2. Experimental Comparison on PFSP-TFT

As follows from Section 6.1, the setting ($NP = 100, \alpha = 0.01, B\_LS$) has been adopted to experimentally compare DEP with the state-of-the-art algorithms for the PFSP with the total flowtime criterion (TFT). Indeed, DEP performances have been compared with those provided in [5] for the four PFSP-TFT state-of-the-art methods: AGA [11], VNS_4 [10], GM-EDA and HGM-EDA [5].

In order to guarantee a fair comparison, the same experimental setting of [5] has been adopted, thus: (i) the performances of DEP have been obtained by running it 20 times for each one of 120 PFSP instances proposed by Taillard [26], and (ii) the experimental results of AGA, VNS, GM-EDA and HGM-EDA have been directly obtained from the additional material provided with [5].

Separately for each $n \times m$ problem configuration, a statistical comparison of the competitor algorithms has been performed by executing the non-parametric Friedman test on the ARPD values obtained by the various algorithms on the different instances of the same $n \times m$ configuration. Then, as suggested in [28], the Finner post-hoc procedure has been adopted to compare the performance differences of DEP with respect to each one of the other competitor. The commonly adopted significance level of 0.05 has been employed.

The best TFT values and the ARPDs of every competitor algorithm are reported in Table 4. The best TFTs in bold indicate when DEP reaches the best value in at least one execution, while the asterisk denotes when it is a new known optimal TFT with respect to [5]. Moreover, for each problem instance, the minimal ARPDs among all the competitors are indicated in bold. Finally, for each problem configuration, the Friedman average ranks of all the algorithms are also provided. The average ranks in bold denote that DEP significantly outperforms the algorithm, while values in italic denote that DEP is significantly outperformed by the algorithm.

The results show that, in 79 instances over 120, DEP reaches the best TFT, and, most remarkably, in 45 cases they are the new known best values with respect to [5]. Moreover, it is worth to notice that DEP has obtained new optimal values for 23 (over 30) instances of size $100 \times m$ and for 18 (over

20) instances of size $200 \times m$, which are reputed to be difficult.

The robustness of DEP is proved by the fact that it presents the lowest ARPD results in 96 instances over 120. Again, in almost all 100 and 200 jobs problems, DEP is the best algorithm in average.

Except the case of 500 jobs, DEP has always the lowest Friedman average rank. The results can be summarized as follows:

- For problems with 20 jobs, all the algorithms perform the same, except GM-EDA which is significantly worse. This may suggest that the best values obtained are probably the optimal values for these instances.
- For problems with 50 jobs, DEP has the lowest average rank values and is significantly better than VNS_4, GM-EDA and HGM-EDA.
- For problems with 100 jobs, DEP has average rank values very close to 1 and it has no clear competitor.
- A similar behaviour is found for problems with 200 jobs, but HGM-EDA, although having a worse average rank and obtaining only two best values over 20, is not significantly worse than DEP in average.
- The only weakness for DEP is found in problems with 500 jobs, where it is outperformed by AGA and VNS_4. This is probably due to a very slow convergence of the DEP population, corroborated by the observation that very few or even zero restart operations were performed.

The conclusion of this analysis is that DEP can be considered among the state-of-the-art PFSP-TFT algorithms and it is the best one on the majority of the benchmark problems.

6.3. Experimental Comparison on PFSP-Makespan

As follows from Section 6.1, the setting ($NP = 20, \alpha = 0.01, L\_LS$) has been adopted to experimentally compare DEP with the state-of-the-art algorithms for the PFSP with the makespan criterion. Indeed, DEP performances have been compared with the state-of-the-art method for PFSP-Makespan, i.e., the iterated greedy algorithm (IG) described in [6], and with GM-EDA and HGM-EDA [5] that, like DEP, navigate (though in a different way) the metric space induced by Kendall-τ distance.

The same experimental setting and statistical analysis of Section 6.2 has been adopted also for PFSP-Makespan. Moreover, it worths to note that, while the experimental results of GM-EDA and HGM-EDA have been directly obtained from the additional material provided with [5], IG has been implemented and executed by following the description provided in [6].

The best makespan values and the ARPDs of every competitor algorithm are reported in Table 5. The best makespan values in bold indicate when DEP reaches the best value in at least one execution, while the asterisk denotes when it is a new known optimal makespan with respect to [5]. Moreover, for each problem instance, the minimal ARPDs among all the competitors are indicated in bold. Finally, for each problem configuration, the Friedman average ranks of all the algorithms are also provided. The average ranks in bold denote that DEP significantly outperforms the algorithm, while values in italic denote that DEP is significantly outperformed by the algorithm.

Table 5 show that DEP reaches the best makespan in 85 instances over 120. Moreover, the robustness of DEP is even better, since it presents the lowest ARPDs in 91 instances.

Differently from TFT, the results are here discussed by aggregating on the number of machines:

- For the problems with 5 machines, DEP has one of the lowest Friedman's average rank and its performances are significantly better than those of GM-EDA, while are comparable with respect to IG and HGM-EDA.
- For the problems with 10 machines, DEP performances, although very good on 20×10 and 50×10 instances, looks to degrade increasing the number of jobs, at least with respect to HGM-EDA.
- The best performances of DEP are found on the 20 machines problems. Here, DEP has always the lowest Friedman's average rank and, most remarkably, it reaches the best ARPD value on all the 30 instances with 100, 200, and 500 jobs.

Summarizing, the performances of DEP seems to increase with the number of machines. Since, for the makespan criterion, the number and the size of fitness plateaus plausibly decrease with the num-

ber of machines, the results seem to indicate that DEP performs well with fitness landscape that are not too much neutral.

6.4. Differential Mutation Analysis

Though, for the permutation flowshop scheduling problem, the proposed composition of genetic operators is original by itself, an additional experimental investigation has been conducted in order to analyze the impact of the main component of DEP, i.e., the discrete differential mutation operator introduced in Section 4.

The experimental analysis has been performed by comparing the original DEP algorithm proposed in this work (i.e., equipped with the differential mutation operator previously described) with three modified DEP schemes where the differential mutation has been replaced by alternative mutation schemes as follows:

- DEP-CRM (completely random mutation): every generated mutant is completely random;
- DEP-RIM (random individual mutation): a mutant individual is randomly selected from the current population;
- DEP-PR (path relinking): a mutant is generated by means of a path relinking procedure between two randomly selected population individuals.

Regarding DEP-PR, the general path relinking procedure [36] for two permutations π_1 and π_2 works by: (i) obtaining the sequence of moves that allows to transform π_1 in π_2, (ii) select a prefix of this sequence, and (iii) apply the prefix sequence to π_1 in order to obtain a new permutation ν somehow in the middle between π_1 and π_2. Actually, in DEP-PR we have chosen the swaps between adjacent permutation elements as moves, thus the adopted path relinking operator reduces to a specification of the differential mutation definition, i.e.,

$$\nu_i = \pi_{r_1} \circ \left(F \cdot \left(\pi_{r_1}^{-1} \circ \pi_{r_2}\right)\right) \qquad (11)$$

where, analogously to equation (6), $r_1, r_2 \in [1, NP]$ are two randomly chosen population indexes different between them and with respect to i. Also in this case the parameter $F \in]0, 1]$ has been self-adapted as described in Section 5.2.

The experimental investigation has been conducted on the first Taillard instance of every problem configuration with $n \in \{50, 100\}$ and $m \in \{5, 10, 20\}$. The proposed DEP scheme with the original differential mutation operator (from now on indicated with DEP-DM) and the three described variants have been run 20 times per instance. The DEP parameters have been set according to Section 6.1. The ARPDs relative to this set of executions are provided in Tables 6 and 7 respectively for PFSP-TFT and PFSP-Makespan. The best ARPDs for each instance are indicated in bold.

Table 6
Mutation Operators Comparison on PFSP-TFT

$n \times m$	DEP-DM	DEP-CRM	DEP-RIM	DEP-PR
50×5	**0.00**	0.93	0.55	0.07
50×10	**0.18**	1.63	1.27	0.54
50×20	**0.09**	1.48	1.09	0.46
100×5	**0.05**	1.45	0.81	0.25
100×10	**0.16**	2.18	1.22	0.78
100×20	**0.37**	3.07	1.78	1.39

Table 7
Mutation Operators Comparison on PFSP-Makespan

$n \times m$	DEP-DM	DEP-CRM	DEP-RIM	DEP-PR
50×5	**0.00**	**0.00**	**0.00**	**0.00**
50×10	**0.07**	0.29	0.16	0.11
50×20	**0.36**	0.93	0.69	0.39
100×5	**0.00**	**0.00**	**0.00**	**0.00**
100×10	**0.10**	0.56	0.13	**0.10**
100×20	**0.06**	3.19	0.79	0.70

Table 6 clearly show that DEP-DM is always the best algorithm on PFSP-TFT. Table 7 confirms that the same is true also for PFSP-Makespan. However, for the makespan objective: on the 5 machines problems all the executions of every mutation variant reach the same final makespan value, and DEP-PR has the same ARPD of DEP-DM on the 100×10 instance, though heavily degrading its performances on the 100×20 instance, thus indicating that the trend observed in Section 6.3 (where DEP looks to be more performant on the 20 machines problems) is probably due to the differential mutation operator.

Summarizing, the general trend shown by the experimental results provided in Tables 6 and 7 is that the four mutation variants of DEP can be ranked with a strong order of preference: 1-st DEP-

DM, 2-nd DEP-PR, 3-rd DEP-RIM, 4-th DEP-CRM.

A plausible explanation of this results is provided in the following. The poor results obtained by DEP-CRM clearly show that the chosen crossover operator, though purposely designed for permutation flowshop problems, is not able to reach good performances by itself because it is not enough "exploitative". Conversely, an exploitative behavior is found in DEP-RIM where a mutant is generated by random selection from the current (thus, evolved) population. However, though DEP-RIM improves DEP-CRM performances, the random individual mutation greatly increases the probability of performing a crossover between two equal genotypes, thus it strongly accelerates the lost of population diversity during the evolution. This premature convergence is mitigated in DEP-PR and DEP-DM that both exploit the current population to generate a mutant individual ν with the additional property that ν is (possibly) not contained in the population. However, the path relinking of DEP-PR constrains ν to be in the union of the metric intervals of every couple of population individuals

$$\nu \in \bigcup_{1 \leq i,j \leq NP} [\pi_i, \pi_j]_K$$

where $[\pi_i, \pi_j]_K$ is the metric interval defined in equation (7). Conversely, it is easy to see that the differential mutation of DEP-DM has not such a restriction. Therefore, by also considering that DEP-DM outperforms the other mutation variants of DEP, the differential mutation operator introduced in this work seems to have a very good trade-off between exploration and exploitation.

6.5. Execution Times

For every algorithm considered in our experimental session we have registered the average CPU time over 10 executions spent to perform the number of allowed fitness evaluations (see Table 1) 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. Furthermore, while for DEP and IG we have used our implementations, the source code of GM-EDA, HGM-EDA, AGA and VNS_4 have been obtained from the additional material of [5].

Tables 8 and 9 show the average execution times in seconds, respectively for PFSP-TFT and PFSP-Makespan. In order to discuss these results, it is worthwhile to note that the local search methods for PFSP can exploit a speed-up procedure in the fitness computation that allows to drastically decrease their execution times [37,7]. Therefore, the speed-up technique greatly improves the results of VNS_4, IG and AGA. Unfortunately, this procedure is not easily exploitable in the other population based algorithms considered. Indeed, on the TFT criterion, the two local search based methods VNS_4 and AGA are, on average, about ten times faster than DEP. Also on the makespan criterion, IG is, on average, about seven times faster than DEP. Nevertheless, DEP looks to be faster than the other population based schemes, i.e., GM-EDA and HGM-EDA, both on PFSP-TFT and PFSP-Makespan.

Table 8
Execution Times on PFSP-TFT

$n \times m$	AGA	VNS_4	GM-EDA	HGM-EDA	DEP
20 × 5	57	43	526	88	132
20 × 10	117	76	693	215	229
20 × 20	236	186	856	557	354
50 × 5	149	102	1444	237	608
50 × 10	263	236	2398	597	1016
50 × 20	563	484	2856	1577	1345
100 × 5	307	230	5279	1038	1563
100 × 10	535	446	8083	2506	2467
100 × 20	1132	950	7518	3403	3133
200 × 10	830	919	18788	11675	4180
200 × 20	1718	1865	20205	14131	13727
500 × 20	5099	4377	195076	106429	70506
AVG	917	826	21977	11871	8272

Table 9
Execution Times on PFSP-Makespan

$n \times m$	IG	GM-EDA	HGM-EDA	DEP
20 × 5	39	489	123	167
20 × 10	112	671	248	234
20 × 20	210	907	612	379
50 × 5	88	2348	421	1204
50 × 10	209	3007	586	951
50 × 20	440	3173	1512	1330
100 × 5	200	8912	6012	5005
100 × 10	392	11265	3829	3318
100 × 20	876	7710	3705	3119
200 × 10	822	46069	31675	24502
200 × 20	1829	20205	16131	12650
500 × 20	3910	101496	46429	16859
AVG	761	17178	9274	5810

7. Conclusions and Future Works

In this work, a new discrete Differential Evolution algorithm for Permutation spaces (DEP) has been proposed. The main contribution is the differential mutation operator which is defined by means of a randomized bubble sort algorithm and brings the most important properties of classical DE to the permutations space. Moreover, a randomly biased selection operator that allows to improve the population diversity in order to mitigate the super-individual effect has been proposed.

Experiments were held on the permutation flowshop scheduling problem (PFSP), both for the total flowtime (TFT) and makespan criteria. The experimental results on PFSP-TFT show that DEP outperforms the other state-of-the-art algorithms and found 45 new optimal solutions with respect to [5]. Also the experiments on PFSP-Makespan show very good results for DEP and give the indication that DEP performances degrade when the landscape neutrality increases.

Promising lines of research for further improvements will focus on the the application of the DEP algorithm to other permutation-based problems (like TSP, QAP, LOP, etc.) and a deeper investigation of DEP performances in relation with the problem landscape properties, Finally we are also planning to implement the discrete differential mutation using generic transpositions and insertions as generating sets.

References

[1] Gupta, J., Stafford, J.E.: Flowshop scheduling research after five decades. European Journal of Operational Research 169, 699–711 (2006)

[2] Gonzalez, T., Sahni, S.: Flowshop and Jobshop Schedules: Complexity and Approximation. Operations Research 26 (1), 36–52 (1978)

[3] Pan, Q.K., Ruiz, R.: A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. Computers and Industrial Engineering 55 (4), 795–816 (2013)

[4] Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. European Journal of Operational Research 165, 479–494 (2005)

[5] Ceberio, J., Irurozki, E., Mendiburu, A., Lozano, J.A. A Distance-based Ranking Model Estimation of Distribution Algorithm for the Flowshop Scheduling Problem. IEEE Transactions on Evolutionary Computation 99, 1–16 (2013)

[6] 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)

[7] Nawaz, M., Enscore Jr., E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA, The International Journal of Management Science 11 (1), 91-95 (1983)

[8] Cheng, V.C., Leung, C.H.C., Liu, J., Milani, A. Probabilistic Aspect Mining Model for Drug Reviews. IEEE Transactions on Knowledge and Data Engineering 99, p.1 (2014)

[9] Franzoni, V., Milani, A. Heuristic semantic walk for concept chaining in collaborative networks. International Journal of Web Information Systems 10 (1), 85–103 (2014)

[10] Costa, W.E., Goldbarg, M.C., Goldbarg, E.G. New VNS heuristic for total flowtime flowshop scheduling problem. Expert Systems with Appl. 39, 8149–8161 (2012)

[11] Xu, X., Xu, Z., Gu, X. An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. Expert Systems with Appl. 38, 7970–7979 (2011)

[12] Liu, J., Reeves, C.R. Constructive and composite heuristic solutions to the $P//\sum C_i$ scheduling problem. European Journal of Operational Research 132, 439–452 (2001)

[13] Storn, R., Price, K. Differential Evolution: A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Jour. of Global Opt. 11, 341–359 (1997)

[14] Milani, A., Santucci, V. Community of scientist optimization: An autonomy oriented approach to distributed optimization. AI Commununications 25, 157-17 (2012)

[15] Baioletti, M., Milani, A., Poggioni, V., Rossi, F. Experimental evaluation of pheromone models in ACOPlan. Ann. Math. Artif. Intell. 62(3-4), 187-217 (2011)

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

[17] Cickova, Z., Stevo, S. Flow Shop Scheduling using Differential Evolution. Management Information Systems 5 (2), 8–13 (2010)

[18] Li, X., Yin, M. An opposition-based differential evolution algorithm for permutation flow shop scheduling based on diversity measure. Adv. in Eng. Soft. 55, 10–31 (2013)

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

[20] Rothlauf, F. Representations for Genetic and Evolutionary Algorithms. Springer, Berlin (2006)

[21] Moraglio, A., Poli, R.: Geometric crossover for the permutation representation. Intelligenza Artificiale 5 (1), 49–63 (2011)

[22] Schiavinotto, T., Stutzle, T. A review of metrics on permutations for search landscape analysis. Computers & Oper. Res. 34 (10), 3143–3153 (2007)

[23] Fisher, S., Yates, F. Statistical tables. Oliver & Boyd, Edinburgh (1943)

[24] Brest, J., Boskovic, B., Mernik, M., Zumer, V. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Trans. on Evol. Comp. 10 (6), 646–657 (2006)

[25] Murata, T., Ishibuchi, H., Tanaka, H. Genetic algorithms for flowshop scheduling problems. Computers & Ind. Eng. 30 (4), 1061–1071 (1996)

[26] Taillard, E. Benchmarks for basic scheduling problems. European Jour. of Oper. Res. 64 (2), 278–285 (1993)

[27] Stützle, T.: Applying iterated local search to the permutation flow-shop problem. Tech. Rep. AIDA-98-04, FG Intellektik, TU Darmstadt (1998)

[28] Derrac, J., Garcia, S., Molina, D., Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1, 3–18 (2011)

[29] H. H. Hoos and T. Stutzle, "Stochastic Local Search: Foundations and Applications," Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[30] I. N. Herstein, "Topics in Algebra," 2nd EditionJohn, Wiley & Sons, 1975.

[31] Rajendran, C., Ziegler, H, "A Multi-Objective Ant-Colony Algorithm for Permutation Flowshop Scheduling to Minimize the Makespan and Total Flowtime of Jobs," In: Computational Intelligence in Flow Shop and Job Shop Scheduling Studies in Computational Intelligence Volume 230, pp. 53–99, 2009.

[32] Mokotoff, E. "Multi-objective Simulated Annealing for Permutation Flow Shop Problems," In: Computational Intelligence in Flow Shop and Job Shop Scheduling Studies in Computational Intelligence Volume 230, pp. 101–150, 2009.

[33] Lin, S. W., Ying, K. C., "Minimizing makespan and total flowtime in permutation flowshops by a bi-objective multi-start simulated-annealing algorithm," Computers & Operations Research, vol. 40, issue 6, pp. 1625-1647, 2013.

[34] Minella, G., Ruiz, R., Ciavotta, C. "A Review and Evaluation of Multiobjective Algorithms for the Flowshop Scheduling Problem," INFORMS Journal on Computing, vol. 20, issue 3, pp. 451–471, 2008.

[35] M. Birattari, "Tuning Metaheuristics: A Machine Learning Perspective," Studies in Computational Intelligence, vol. 197, Springer-Verlag, Berlin, Germany, 2009.

[36] F. Glover, M. Laguna, and R. Marti. "Fundamentals of scatter search and path relinking," Control and Cybernetics, vol. 39, pp. 653-684, 2000.

[37] Li X, Wang Q, Wu C. "Efficient composite heuristics for total flowtime minimization in permutation flow shops," OMEGA journal, vol. 37, pp. 155–164, 2009.

Acknowledgments

This work has been held within the activities of the project Mobile Knowledge Agents in Evolutionary Environments of the Laboratory for Knowledge and Information Technology (KITLab) at the Department of Mathematics and Computer Science, University of Perugia. Moreover, experimental results presented in this paper has been possible thanks to the grid resources and services provided by the European Grid Infrastructure (EGI), the Italian Grid Infrastructure (IGI) and the National Grid Initiatives that support the Virtual Organization (VO) COMPCHEM.

Table 4
Experimental Results for Total Flowtime Criterion

Instance	Best TFT	AGA	VNS_4	GM-EDA	HGM-EDA	DEP	Instance	Best TFT	AGA	VNS_4	GM-EDA	HGM-EDA	DEP
20×5	**14033**	**0.00**	**0.00**	0.18	**0.00**	**0.00**	100×5	*__253605__	0.29	1.25	0.87	0.23	**0.05**
	15151	**0.00**	**0.00**	0.48	**0.00**	**0.00**		*__242579__	0.30	1.80	1.08	0.35	**0.05**
	13301	**0.00**	**0.00**	0.50	**0.00**	**0.00**		*__238075__	0.22	1.49	0.85	0.26	**0.07**
	15447	**0.00**	**0.00**	0.43	**0.00**	**0.00**		227889	0.17	1.29	0.78	0.20	**0.06**
	13529	**0.00**	**0.00**	0.21	**0.00**	**0.00**		240589	0.21	1.29	0.80	0.23	**0.02**
	13123	**0.00**	**0.00**	0.08	**0.00**	**0.00**		*__232689__	0.32	1.52	0.90	0.28	**0.06**
	13548	**0.00**	**0.00**	0.79	**0.00**	**0.00**		240669	**0.15**	1.34	1.00	0.34	0.25
	13948	**0.00**	**0.00**	0.18	**0.00**	**0.00**		*__231064__	0.29	1.79	1.06	0.35	**0.07**
	14295	**0.00**	**0.00**	0.18	**0.00**	**0.00**		*__248039__	0.40	1.66	1.05	0.38	**0.09**
	12943	**0.00**	**0.00**	0.46	**0.00**	**0.00**		*__243258__	0.19	1.44	1.00	0.28	**0.07**
Avg Rank		2.5	2.5	**5**	2.5	2.5	Avg Rank		2.2	**5**	**4**	**2.7**	1.1
20×10	**20911**	**0.00**	**0.00**	0.45	**0.00**	**0.00**	100×10	*__299101__	0.43	1.63	1.80	0.44	**0.16**
	22440	**0.00**	**0.00**	0.54	**0.00**	**0.00**		*__274566__	0.60	1.58	2.08	0.69	**0.28**
	19833	**0.00**	**0.00**	0.31	**0.00**	**0.00**		*__288543__	0.37	1.57	1.74	0.38	**0.18**
	18710	**0.00**	**0.00**	0.75	**0.00**	**0.00**		*__301552__	0.50	1.79	2.08	0.53	**0.18**
	18641	**0.00**	**0.00**	0.35	**0.00**	**0.00**		*__284722__	0.61	1.64	1.95	0.54	**0.22**
	19245	**0.00**	**0.00**	0.77	**0.00**	**0.00**		*__270483__	0.42	1.76	1.83	0.45	**0.19**
	18363	**0.00**	**0.00**	0.47	**0.00**	**0.00**		*__280257__	0.37	1.58	1.65	0.40	**0.25**
	20241	**0.00**	**0.00**	0.47	**0.00**	**0.00**		*__291231__	0.49	1.77	2.03	0.61	**0.27**
	20330	**0.00**	**0.00**	0.27	**0.00**	**0.00**		302624	0.36	1.46	1.76	0.41	**0.20**
	21320	**0.00**	**0.00**	0.24	**0.00**	**0.00**		*__291705__	0.48	1.84	1.68	0.50	**0.06**
Avg Rank		2.5	2.5	**5**	2.5	2.5	Avg Rank		2.1	**4.1**	**4.9**	**2.9**	1
20×20	**33623**	**0.00**	**0.00**	0.65	**0.00**	**0.00**	100×20	*__366438__	0.80	1.70	2.26	0.67	**0.37**
	31587	**0.00**	**0.00**	0.28	**0.00**	**0.00**		*__373138__	0.55	1.43	2.04	0.58	**0.25**
	33920	**0.00**	**0.00**	0.04	**0.00**	**0.00**		371417	0.47	1.31	1.93	0.36	**0.21**
	31661	**0.00**	**0.00**	0.28	**0.00**	**0.00**		*__373574__	0.60	1.36	1.92	0.45	**0.26**
	34557	**0.00**	**0.00**	0.26	**0.00**	**0.00**		*__369903__	0.57	1.35	1.92	0.47	**0.19**
	32564	**0.00**	**0.00**	0.30	**0.00**	**0.00**		*__372752__	0.51	1.46	2.17	0.42	**0.30**
	32922	**0.00**	**0.00**	0.61	**0.00**	**0.00**		*__373447__	0.70	1.82	2.19	0.63	**0.33**
	32412	**0.00**	**0.00**	0.52	**0.00**	**0.00**		385456	0.46	1.41	1.96	0.43	**0.20**
	33600	**0.00**	**0.00**	0.56	**0.00**	**0.00**		*__375352__	0.62	1.52	2.01	0.52	**0.41**
	32262	**0.00**	**0.00**	0.41	**0.00**	**0.00**		379899	0.48	1.29	2.05	0.49	**0.46**
Avg Rank		2.5	2.5	**5**	2.5	2.5	Avg Rank		**2.8**	**4**	**5**	2.2	1
50×5	64803	**0.05**	0.78	0.79	0.12	**0.05**	200×10	1047662	0.48	1.25	1.19	**0.17**	0.21
	68062	**0.06**	0.88	0.94	0.12	0.08		*__1035783__	0.94	1.54	1.49	0.32	**0.15**
	63162	**0.19**	1.21	1.34	0.38	0.21		*__1045706__	0.66	1.62	1.30	0.32	**0.15**
	68226	0.17	1.12	1.27	0.22	**0.13**		*__1029580__	0.77	1.65	1.38	0.45	**0.12**
	69392	**0.09**	0.87	0.89	0.15	**0.09**		*__1036464__	0.68	1.35	1.37	0.19	**0.13**
	66841	0.10	0.80	0.82	0.18	**0.04**		1006650	0.50	1.36	1.39	**0.19**	0.23
	66258	0.03	0.74	0.95	0.07	**0.02**		*__1052786__	0.95	1.66	1.23	0.24	**0.10**
	64359	**0.05**	0.89	0.97	0.23	**0.05**		*__1044961__	0.62	1.51	1.39	0.25	**0.11**
	62981	0.09	0.83	0.81	0.14	**0.05**		*__1023315__	0.81	1.61	1.29	0.28	**0.24**
	*__68843__	0.15	1.13	1.01	0.29	**0.10**		*__1029198__	0.97	1.87	1.48	0.39	**0.25**
Avg Rank		1.6	**4.2**	**4.8**	**3**	1.4	Avg Rank		**3**	**4.8**	**4.2**	1.8	1.2
50×10	*__87204__	0.33	1.12	2.11	0.39	**0.18**	200×20	*__1225817__	0.72	1.44	1.68	0.34	**0.16**
	82820	**0.22**	1.09	2.45	0.60	0.30		*__1239246__	1.07	1.67	1.66	0.54	**0.21**
	79987	0.23	1.07	1.84	0.36	**0.22**		*__1263134__	1.08	1.65	1.57	0.48	**0.26**
	*__86545__	0.21	0.94	1.87	0.36	**0.16**		*__1233443__	1.25	1.84	1.73	0.58	**0.24**
	86450	**0.14**	0.90	2.02	0.38	0.25		*__1220117__	1.12	1.79	1.93	0.53	**0.17**
	86637	0.13	0.77	1.55	0.29	**0.11**		*__1223238__	1.17	1.69	1.69	0.46	**0.19**
	88866	**0.25**	0.89	1.97	0.48	0.42		*__1237116__	1.03	1.65	1.66	0.64	**0.15**
	*__86820__	0.19	0.95	2.04	0.36	**0.01**		*__1238975__	1.25	1.72	1.72	0.51	**0.19**
	85526	0.29	1.11	2.10	0.42	**0.28**		*__1225186__	1.44	1.91	1.80	0.59	**0.14**
	88077	**0.09**	0.76	2.00	0.45	0.42		*__1244200__	1.16	1.62	1.68	0.52	**0.11**
Avg Rank		1.6	**4**	**5**	**3**	1.4	Avg Rank		**3**	**4.5**	**4.5**	2	1
50×20	125831	**0.10**	0.65	1.76	0.39	0.14	500×20	6708053	**0.11**	0.35	8.90	2.02	1.00
	119259	**0.04**	0.51	1.58	0.22	0.06		6829668	**0.25**	0.38	8.58	1.94	0.66
	116459	**0.19**	0.73	2.24	0.44	0.28		6747387	**0.24**	0.41	8.46	2.04	1.07
	120712	**0.22**	0.61	1.92	0.34	0.34		6787054	**0.26**	0.45	8.75	1.89	0.84
	118184	0.40	0.86	2.30	0.52	**0.39**		6755257	**0.39**	0.41	8.72	1.92	0.74
	120703	0.19	0.62	1.78	0.35	**0.16**		6751496	**0.19**	0.42	8.58	2.13	0.32
	122962	0.38	0.71	2.10	0.47	**0.36**		6708860	**0.27**	0.45	9.15	2.05	0.93
	122489	0.16	0.75	2.24	0.55	**0.14**		6769821	**0.31**	0.58	8.62	2.09	0.73
	121872	0.16	0.76	1.79	0.37	**0.12**		6720474	**0.15**	0.46	8.69	1.91	0.96
	124064	**0.23**	0.90	1.95	0.42	0.29		6767645	**0.19**	0.44	8.51	2.00	0.86
Avg Rank		1.5	**4**	**5**	**3**	1.5	Avg Rank		*1*	*2.1*	**5**	**4**	2.9

Table 5
Experimental Results for Makespan Criterion

Instance	Best Ms.	IG	GM-EDA	HGM-EDA	DEP	Instance	Best Ms.	IG	GM-EDA	HGM-EDA	DEP
20×5	**1278**	**0.00**	**0.00**	**0.00**	**0.00**	100×5	**5493**	**0.00**	0.04	**0.00**	**0.00**
	1359	**0.00**	0.07	**0.00**	**0.00**		5268	0.13	0.32	**0.00**	0.13
	1081	**0.00**	0.37	**0.00**	**0.00**		**5175**	**0.00**	0.25	**0.00**	**0.00**
	1293	**0.00**	0.15	**0.00**	**0.00**		5014	0.08	0.24	**0.00**	0.06
	1235	**0.00**	0.08	**0.00**	**0.00**		**5250**	**0.00**	0.10	**0.00**	**0.00**
	1195	**0.00**	1.09	**0.00**	**0.00**		**5135**	**0.00**	0.19	**0.00**	**0.00**
	1234	0.31	1.38	**0.24**	0.41		**5246**	**0.00**	0.23	**0.00**	**0.00**
	1206	**0.00**	0.17	**0.00**	**0.00**		**5094**	0.01	0.22	**0.00**	**0.00**
	1230	**0.00**	0.57	**0.00**	**0.00**		**5448**	**0.00**	0.46	**0.00**	**0.00**
	1108	**0.00**	**0.00**	**0.00**	**0.00**		5322	0.06	0.30	**0.00**	0.04
Avg Rank		2.1	**3.7**	2	2.2	Avg Rank		**2.45**	**4**	1.6	1.95
20×10	**1582**	**0.00**	0.51	**0.00**	**0.00**	100×10	5770	0.19	0.78	**0.00**	0.10
	1659	**0.00**	0.66	**0.00**	**0.00**		5349	0.30	0.71	**0.06**	0.24
	1496	**0.00**	1.14	**0.00**	**0.00**		5676	0.22	0.07	**0.04**	0.05
	1377	**0.00**	1.02	**0.00**	**0.00**		5781	1.05	1.16	**0.24**	0.66
	1419	**0.00**	0.49	**0.00**	**0.00**		5467	0.91	0.77	**0.29**	0.52
	1397	**0.00**	0.50	**0.00**	**0.00**		**5308**	0.12	0.32	**0.00**	**0.00**
	1484	**0.00**	0.27	**0.00**	**0.00**		5596	0.30	0.79	**0.04**	0.12
	1538	0.26	0.91	0.07	**0.00**		5623	0.54	0.92	**0.23**	0.48
	1593	**0.00**	1.13	**0.00**	**0.00**		5875	0.74	0.87	**0.00**	0.48
	1591	0.09	1.01	**0.00**	**0.00**		5848	0.91	0.75	**0.00**	0.54
Avg Rank		2.2	**4**	1.95	1.85	Avg Rank		**3.3**	**3.7**	*1.05*	1.95
20×20	**2297**	0.02	18.94	**0.00**	**0.00**	100×20	**6245**	2.18	2.50	0.58	**0.06**
	2099	**0.00**	35.59	**0.00**	**0.00**		**6210**	2.34	1.84	0.66	**0.13**
	2326	0.09	12.98	**0.00**	**0.00**		**6303**	2.04	1.87	0.44	**0.07**
	2223	**0.00**	24.52	**0.00**	**0.00**		6291	1.60	1.54	0.48	**0.23**
	2291	0.12	25.01	0.04	**0.00**		**6362**	1.82	2.12	0.36	**0.05**
	2226	0.09	27.27	**0.00**	**0.00**		**6423**	1.85	1.59	0.48	**0.02**
	2273	**0.00**	20.37	**0.00**	**0.00**		**6298**	2.39	1.73	0.54	**0.10**
	2200	0.15	22.82	**0.00**	**0.00**		**6423**	2.76	2.52	0.97	**0.15**
	2237	**0.00**	14.48	**0.00**	**0.00**		**6292**	2.52	2.10	0.81	**0.17**
	2178	0.08	27.73	**0.00**	**0.00**		**6476**	1.47	1.64	0.36	**0.06**
Avg Rank		**2.6**	**4**	1.75	1.65	Avg Rank		**3.7**	**3.3**	**2**	1
50×5	**2724**	**0.00**	0.29	**0.00**	**0.00**	200×10	10872	0.16	0.33	**0.00**	0.31
	2834	0.10	0.42	**0.00**	**0.00**		10493	0.47	0.45	**0.03**	0.79
	2621	**0.00**	0.27	**0.00**	**0.00**		10922	0.38	0.81	**0.00**	0.68
	2751	**0.00**	0.62	**0.00**	**0.00**		10889	1.17	0.26	**0.01**	0.20
	2863	**0.00**	0.03	**0.00**	**0.00**		10527	0.14	0.09	**0.00**	0.15
	2829	**0.00**	0.14	**0.00**	**0.00**		10330	0.35	0.51	**0.01**	0.45
	2725	**0.00**	0.40	**0.00**	**0.00**		10857	0.27	0.52	**0.00**	0.47
	2683	**0.00**	0.71	**0.00**	**0.00**		10731	0.26	0.62	**0.02**	0.61
	2552	0.16	0.35	**0.00**	0.06		10438	0.26	0.37	**0.03**	0.27
	2782	**0.00**	**0.00**	**0.00**	**0.00**		10676	0.49	0.52	**0.02**	0.51
Avg Rank		2.25	**3.85**	1.9	2	Avg Rank		*2.4*	3.5	*1*	3.1
50×10	**3025**	0.46	1.22	**0.00**	**0.00**	200×20	**11243**	1.98	1.46	0.56	**0.24**
	2877	1.47	1.81	0.45	**0.33**		**11269**	2.71	1.59	0.40	**0.09**
	2852	0.91	1.54	0.42	**0.32**		**11397**	1.72	1.32	0.30	**0.02**
	3063	0.42	0.85	**0.00**	**0.00**		**11345**	2.22	1.33	0.31	**0.04**
	2979	1.31	1.54	**0.57**	0.88		**11293**	1.71	1.37	0.28	**0.10**
	3006	0.91	1.73	0.03	**0.00**		**11234**	1.90	1.43	0.57	**0.11**
	3098	0.88	2.07	**0.26**	0.38		**11424**	1.58	1.16	0.23	**0.04**
	3038	0.34	0.79	**0.10**	0.20		**11402**	2.06	1.11	0.29	**0.13**
	2900	0.59	1.14	0.07	**0.06**		**11241**	2.35	1.61	0.70	**0.23**
	3078	1.66	1.40	0.16	**0.00**		**11339**	2.21	1.19	0.35	**0.04**
Avg Rank		**3.1**	**3.9**	1.6	1.4	Avg Rank		**2**	**4**	**3**	1
50×20	3870	1.19	2.14	**0.49**	0.59	500×20	**26182**	1.15	3.38	0.23	**0.02**
	3711	1.96	3.18	0.27	**0.08**		**26716**	0.83	2.77	0.18	**0.05**
	3658	2.09	3.03	0.55	**0.10**		**26494**	0.83	3.71	0.31	**0.10**
	3739	1.14	2.09	0.43	**0.12**		26558	1.12	3.19	**0.15**	0.12
	3625	1.75	1.93	**0.39**	0.40		**26379**	0.93	2.30	0.28	**0.04**
	3698	1.47	2.54	0.32	**0.27**		**26581**	0.78	2.99	0.12	**0.01**
	3720	1.60	2.23	0.30	**0.15**		**26424**	0.77	3.17	0.45	**0.21**
	3712	2.11	2.45	0.43	**0.22**		**26646**	0.89	2.64	0.33	**0.00**
	3754	1.45	2.21	0.51	**0.37**		**26123**	1.01	3.08	0.28	**0.11**
	3768	1.65	2.65	0.40	**0.07**		**26551**	0.88	3.14	0.21	**0.06**
Avg Rank		**3**	**4**	**1.8**	1.2	Avg Rank		**3**	**4**	**2**	1