Linear Ordering Optimization with a Combinatorial Differential Evolution

Marco Baioletti, Alfredo Milani, Valentino Santucci
Department of Mathematics and Computer Science
University of Perugia
Perugia, Italy
{marco.baioletti,alfredo.milani}@unipg.it, valentino.santucci@dmi.unipg.it

***Abstract*—In this work, the Linear Ordering Problem (LOP) has been approached using a discrete algebraic-based Differential Evolution for the Linear Ordering Problem (LOP). The search space of LOP is composed by permutations of objects, thus it is possible to use some group theoretical concepts and methods. Indeed, the proposed algorithm is a combinatorial Differential Evolution scheme designed by exploiting the group structure of the LOP solutions in order to mimic the classical Differential Evolution behavior observed in continuous spaces. In particular, the proposed differential mutation operator allows to obtain both scaled and extended differences among LOP solutions represented by permutations. The performances have been evaluated over widely known LOP benchmark suites and have been compared to the state-of-the-art results.**

***Index Terms*—Differential Evolution, Linear Ordering Problem, Combinatorial Optimization**

I. Introduction

The Linear Ordering Problem (LOP) is a classical NP-Hard combinatorial optimization problem [1], [2] that has received considerable attention because of its many applications in diverse research fields such as economy [3], graph theory [4], archeology [5] and computational social choice [6].

LOP can be straightforwardly formulated as a matrix triangulation problem [7]. Given a $n \times n$ matrix H, LOP requires to find a permutation π of the row and column indices $\{1, \ldots, n\}$ that maximizes the objective function

$$f(\pi) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} H_{\pi(i),\pi(j)}. \tag{1}$$

Basically, the goal is to find a simultaneous permutation of the rows and columns of H such that the sum of the super-diagonal entries is maximized.

Since LOP is NP-Hard [8], exact methods are able to find optimal solutions only in small problem instances. Anyway, the permutation structure of the LOP solutions allows to apply a variety of meta-heuristics and evolutionary algorithms specifically designed for permutation-based search spaces. Among these, recently, the Differential Evolution for Permutations (DEP) algorithm has been successfully proposed in [9] for the Permutation Flowshop Scheduling Problem (PFSP).

DEP is a completely discrete variant of the popular numerical Differential Evolution (DE) algorithm [10]. The core component of numerical DE is its differential mutation operator that allows to self-adapt DE population to the objective function landscape at hand by exploiting the differences among the solutions in the population [11]. DEP mimics the same behavior of classical DE in the combinatorial space of permutations. Its key idea resides in the definition of the operations of difference, sum, and truncation on the permutations space. These operations are somehow consistent with their usual definitions in the numerical space $\mathbb{R}^n$. This is made possible by exploiting the algebraic structure of the permutations space where its elements, i.e., the permutations, form a group in the algebraic sense, namely, the symmetric group.

Since DEP has obtained state-of-the-art results on the PFSP problem [9], and in order to investigate its applications to other classes of permutation-based combinatorial optimization problems, here we provide a DEP variant for LOP and we analyze its performances on a large set of widely known benchmark problem instances.

Furthermore, with respect to [9], the differential mutation procedure has been expanded in order to produce not only only truncated differences but also extended differences between permutations.

The rest of the paper is organized as follows. In Section II, we provide a review of the state-of-the-art LOP methods found in literature. Section III describes the DEP algebraic-based differential mutation operator by also introducing the new procedure that allows to consider extended differences between permutations. The full DEP scheme adapted for LOP is described in Section IV. An experimental analysis of the proposed approach is provided in Section V. Finally, conclusions are drawn in Section VI where some future lines of research are also depicted.

II. Related Works

Due to the NP-Hardness of LOP, as highlighted in [7], exact methods are able to find optimal solutions only in easy problem instances and dramatically deteriorate their execution time with the increasing of the instance size.

To overcome these drawbacks, constructive heuristics have been proposed with the aim of providing a good enough solution in a reasonable amount of time. Among these there are: the two methods proposed by Becker in [12], the heuristic of Chenery and Watanabe [13], and the construction methods based on insertion moves proposed in [2].

However, heuristic methods, although fast, are not effective and do not find optimal solutions also on easy LOP instances.

For this reason they are often used to feed an initial solution to a local search. In LOP, the most used solutions neighborhoods adopted for the local search methods are the insertion, interchange, and adjacent swap neighborhoods [1].

Local search algorithms are not able to escape from local optima. GRASP [1] turns around this problem by repeatedly restarting a local search every time a local optimum is found. However, more effective solutions are obtained by the meta-heuristic methods. These can be divided in two categories: the ones based on a single incumbent solution, and the population based meta-heuristics.

Among the single-solution methods for LOP there are the CK method [14] and some implementations of popular meta-heuristics like Tabu Search [15], Simulated Annealing [16], Variable Neighborhood Search [17] and Iterated Local Search (ILS) [7]. Population based meta-heuristics exploit a population of solutions in order to direct the search for the optimum. LOP methods falling in this category are implementations of: Scatter Search (SS) [18], Genetic Algorithm (GA) [19], and Memetic Algorithm (MA) [7]. Conversely from GA that only employs genetic operators to evolve the population of solutions, SS and MA can be thought as hybrid methods for their extensive use of local search procedures.

According to [7] and the more recent [20], MA and ILS are to be considered the state-of-the-art algorithms for LOP.

ILS [7], starting from a random solution, iteratively alternates the two phases of local search and perturbation. The local search procedure is based on the insertion neighborhood, while the perturbation is performed by shaking the local optimum with a given number of interchange moves. Moreover, a new local optimum is accepted not only if it improves the incumbent solution, but also if it shows a slight worsening.

MA [7] evolves a small population of distinct local optima by applying a local search procedure to the offspring solutions generated by a crossover operator. While the same local search procedure of ILS is employed, in [7] four different crossover operators have been experimented and the results show that the best one is OBX (order based crossover) [21].

Further improvements to ILS and MA have been proposed in [20] where a theoretical investigation of the LOP insertion neighborhood shows that not all the solution neighbors have to be evaluated in order to select the best one. This approach has the cost of an expensive preliminary computation of the so called "restriction matrix". The resulting variants of ILS and MA are referred, respectively, as ILS_R and MA_R.

Finally, some applications of the Differential Evolution (DE) algorithm to LOP have been proposed in [22], [23], [24]. However, conversely from the approach followed in this work, they adopt the classical DE and rely on the "random key" transformation scheme to decode a numerical vector into a permutation just before the fitness evaluation. This approach brings to a distinction between the phenotypic and genotypic spaces, thus introducing large plateaus in the numerical landscape. This is probably the reason of their poor performances.

III. Differential Mutation for Permutations

Differential Evolution (DE) [10] is a popular and powerful evolutionary algorithm over continuous search spaces using the differential mutation operator as its key component [11]. In the most common variant, for each population individual $x_i \in \mathbb{R}^n$, three different parents $x_{r_0}, x_{r_1}, x_{r_2}$ (also different from x_i) are randomly selected from the current population and a mutant $v_i \in \mathbb{R}^n$ is generated according to

$$v_i = x_{r_0} + F \cdot (x_{r_1} - x_{r_2}) \tag{2}$$

where $F > 0$ denotes the scale factor parameter. It has been argued that the differential mutation confers to DE the "contour matching" property (term coined by Price et al. in [11]), i.e., it allows DE to automatically adapt both mutation step size and orientation to the objective function landscape.

In this section we describe and extend the combinatorial differential mutation scheme previously introduced in [9] for the Permutation Flowshop Scheduling Problem (PFSP). In PFSP, this operator allowed to obtain state-of-the-art results but constrains F to the interval $[0, 1]$. Here, we extend the scheme to allow also $F > 1$.

The permutations of the set $\{1, 2, \dots, n\}$, together with the usual permutations composition operator $\circ$, form a group denoted by $S(n)$ where each $\pi \in S(n)$ has its inverse, denoted by π^{-1}. Moreover, the identity permutation e, defined as $e(i) = i$ for each $1 \le i \le n$, is the identity element of $S(n)$.

As introduced in [9], it is possible to consistently define the sum and difference operators of two elements $\pi_1, \pi_2 \in S(n)$ as:

$$\pi_1 \oplus \pi_2 := \pi_1 \circ \pi_2 \tag{3}$$

$$\pi_1 \ominus \pi_2 := \pi_2^{-1} \circ \pi_1 \tag{4}$$

Now, in order to complete the definition of the mutation operator as in equation (2), we have to introduce the concept of multiplication of a permutation π by a real number F to obtain the "scaled" permutation $F \odot \pi$. Once this operation is defined, we can rewrite the differential mutation of equation (2) for the permutations space as:

$$\nu_i = \pi_{r_0} \circ \left(F \odot \left(\pi_{r_2}^{-1} \circ \pi_{r_1}\right)\right) \tag{5}$$

As discussed in [9], the multiplication of a permutation by a positive real value can be defined by considering a generating set of $S(n)$, i.e., a set $G \subseteq S(n)$ such that every element of $S(n)$ can be written as a compositions chain of some elements of G.

Given a generating set $G \subseteq S(n)$ and any $\pi \in S(n)$, let $\langle g_1, \dots, g_L \rangle$ be a decomposition of π, i.e., $\pi = g_1 \circ \cdots \circ g_L$, where $g_1, \dots, g_L \in G$. Let $k = \lceil F \cdot L \rceil$, then the multiplication of π by $0 \le F \le 1$ is defined as $F \odot \pi := g_1 \circ \cdots \circ g_k$. While, for $F > 1$ it is not possible to exceed the maximum length D of a decomposition in terms of G. Hence, $k = min\{\lceil F \cdot L \rceil, D\}$ and the decomposition of π is extended by adding $\langle g_{L+1}, \dots, g_k \rangle$ such that $\langle g_1, \dots, g_L, g_{L+1}, \dots, g_k \rangle$ is a minimal decomposition of some permutation. Therefore, $F \odot \pi := g_1 \circ \cdots \circ g_L \circ g_{L+1} \circ \cdots \circ g_k = \pi \circ g_{L+1} \circ \cdots \circ g_k$.

Interestingly, the generating set G allows a useful geometric interpretation of the search space. Indeed, it is possible to represent the permutations search space as a Cayley graph, whose vertices are the permutations of $S(n)$ and, for any $\pi \in S(n)$ and $g \in G$, the vertices corresponding to π and $\pi \circ g$ are joined by an arc labeled with g. Using the Cayley graph, it is possible to: (i) define a distance between two permutations $\pi_1, \pi_2 \in S(n)$, corresponding to the length of a shortest path from π_1 to π_2, (ii) compute $\pi_1 \ominus \pi_2$ as the compositions chain of the arcs labels in a shortest path from π_2 to π_1, (iii) interpret the scaled difference as a truncated or an extended shortest path, respectively, for $0 \leq F \leq 1$ or $F > 1$.

Various generating sets for $S(n)$ are possible. Following [9], we have decided to work with the set of simple transpositions $ST = \{(i, i{+}1)_T : 1 \leq i \leq n-1\}$, where $(i, i+1)_T$ denotes the permutation which only swaps the two adjacent elements at positions i and $i+1$. ST has $n-1$ elements and induces decompositions with maximum length of $D = \binom{n}{2}$ (this is also the diameter of the corresponding Cayley graph). By recalling that an inversion of a permutation π is a pair of indices $\{i, j\}$ such that $i < j$ and $\pi(i) > \pi(j)$, the number of inversions $I(\pi)$ of any $\pi \in S(n)$ equals to the length of a minimal decomposition of π in terms of ST. Moreover, ST induces the Kendall-τ distance d_K [25] which, if applied to any $\pi_1, \pi_2 \in S(n)$, is equal to $I(\pi_2^{-1} \circ \pi_1)$.

Now, in order to compute $F \odot \pi$, we need a procedure to obtain a minimal decomposition of π in terms of the simple transpositions. This can be done using the well known bubble sort algorithm by recording the adjacent swaps (i.e., the simple transpositions) performed during the sort. However, π has several different shortest representations as compositions chain of simple transpositions. Hence, in order to design a mutation scheme as fair as possible, we defined a randomized version of bubble sort, here outlined in Algorithm 1.

Algorithm 1 Randomized Bubble Sort

```
1: function RandBS($\pi \in S(n)$)
2:     $CC \leftarrow \langle\ \rangle$
3:     $LST \leftarrow \{i : \pi[i] > \pi[i+1]\}$
4:     while $LST \neq \emptyset$ do
5:         $i \leftarrow$ RemoveRandomElement($LST$)
6:         Swap $\pi[i]$ and $\pi[i+1]$
7:         Append $(i, i+1)_T$ to $CC$
8:         if $i > 0$ and $i-1 \notin LST$ and $\pi[i-1] > \pi[i]$ then
9:             Add $i-1$ to $LST$
10:        end if
11:        if $i < n-1$ and $i+1 \notin LST$ and $\pi[i+1] > \pi[i+2]$ then
12:            Add $i+1$ to $LST$
13:        end if
14:    end while
15:    return reverse($CC$)
16: end function
```

$RandBS$ sorts the permutation π towards the identity permutation e with the optimal number of adjacent swaps. Indeed, LST is initialized with the adjacent inversions of π and at each iteration of the while loop: (i) a simple transposition is chosen from LST and is applied to π, thus reducing by one the number of inversion of π; (ii) the applied simple transposition is stored in CC; (iii) LST is updated with the new adjacent inversion(s) in the current π. The search space diameter bounds the number of iterations to $\binom{n}{2} = O(n^2)$, then the time complexity of $RandBS$ is $O(n^2)$ as the classical Bubble-sort algorithm. Furthermore, by reversing the list CC at the end of its computation, $RandBS$ produces as a list one the minimal-length random decompositions of π in terms of simple transpositions.

It is now possible to define the procedures to compute $F \odot \pi$ for both the cases $0 \leq F \leq 1$ and $F > 1$. These are described in the following subsections.

A. Case $0 \leq F \leq 1$

For the case $0 \leq F \leq 1$, we have to compute a random decomposition of the difference permutation, truncate and then evaluate it. More formally, the differential mutation of equation (5) can be computed, when $F \in [0, 1]$, with the following steps:

1) compute the difference $\delta = \pi_{r_1} \ominus \pi_{r_2} = \pi_{r_2}^{-1} \circ \pi_{r_1}$;
2) perform $RandBS(\delta)$ in order to obtain a random decomposition $\langle (i_1, i_1+1)_T, \dots, (i_L, i_L+1)_T \rangle$ of δ in terms of simple transpositions;
3) compute the scaled difference $\delta^* = F \odot \delta = (i_1, i_1+1)_T \circ \cdots \circ (i_k, i_k+1)_T$ where $k = \lceil F \cdot L \rceil$;
4) obtain the mutant as $\nu = \pi_{r_0} \oplus \delta^* = \pi_{r_0} \circ \delta^*$.

It worths to note that $I(\delta^*) = \lceil F \cdot L \rceil = d_K(\nu, \pi_{r_0})$.

B. Case $F > 1$

The multiplication $F \odot \pi$ for $F > 1$ reduces to: (i) compute the length L of a minimal decomposition of π, and (ii) apply $min\{\lceil F \cdot L \rceil, D\} - L$ simple transpositions to π in a way that $I(F \odot \pi) = min\{\lceil F \cdot L \rceil, D\}$. This guarantees that a minimal decomposition of $F \odot \pi$ contains a minimal decomposition of π as prefix.

Operatively, $F \odot \pi$ can be obtained by running $RandBS$ on the reverse of π, that is, on $\pi^R = \langle \pi(n), \pi(n-1), \dots, \pi(1) \rangle$. Indeed, if a minimal decomposition of π^R has length L^R, then a minimal decomposition of π has length $L = D - L^R$. Since the pair of indices $\{i, j\}$ is an inversion of π^R if and only if $\{n{+}1{-}j, n{+}1{-}i\}$ is not an inversion of π, the random decomposition $\langle (i_1, i_1+1)_T, \dots, (i_{L^R}, i_{L^R}+1)_T \rangle$ of π^R, produced by $RandBS(\pi^R)$, can be easily transformed to the sequence $s = \langle (n-i_1, n+1-i_1)_T, \dots, (n-i_{L^R}, n+1-i_{L^R})_T \rangle$. The sequence s has the important property that every prefix p of s produces a permutation σ such that $I(\sigma) = I(\pi) + length(p)$.

Therefore, for $F > 1$, the differential mutation of equation (5) can be computed using the following steps:

1) compute the difference $\delta = \pi_{r_1} \ominus \pi_{r_2} = \pi_{r_2}^{-1} \circ \pi_{r_1}$;
2) reverse δ in δ^R;
3) perform $RandBS(\delta^R)$ in order to obtain the random decomposition $\langle (i_1, i_1+1)_T, \dots, (i_{L^R}, i_{L^R}+1)_T \rangle$ of δ^R in terms of simple transpositions;
4) let $L = D - L^R$ then $k = min\{\lceil F \cdot L \rceil, D\}$;
5) compute the extended difference δ^* by composing δ with the transformation of the first $k-L$ simple transpositions in the sequence above, i.e., $\delta^* = F \odot \delta = \delta \circ (n-i_1, n+1-i_1)_T \circ \cdots \circ (n-i_{k-L}, n+1-i_{k-L})_T$;
6) obtain the mutant as $\nu = \pi_{r_0} \oplus \delta^* = \pi_{r_0} \circ \delta^*$.

Finally, it worths to note that, analogously to the case $0 \leq F \leq 1$, $I(\delta^*) = \lceil F \cdot L \rceil = d_K(\nu, \pi_{r_0})$.

IV. Differential Evolution for LOP

The Differential Evolution for the LOP Permutations space (DEP), outlined in Algorithm 2, directly evolves a population of NP permutations $\pi_1, \ldots, \pi_{NP}$. Its main scheme resembles that of the classical DE with the internal modification of its genetic operators and the introduction of a restart mechanism. All the DEP components are described in the following.

Algorithm 2 Differential Evolution for Permutations

```
1:  Initialize Population
2:  while evaluations budget is not exhausted do
3:      for $i \leftarrow 1$ to $NP$ do
4:          $\nu_i \leftarrow$ DifferentialMutation($i, F$)
5:          $\upsilon_i^{(1)}, \upsilon_i^{(2)} \leftarrow$ Crossover($\pi_i, \nu_i, CR$)
6:          Evaluate $f(\upsilon_i^{(1)})$ and $f(\upsilon_i^{(2)})$
7:      end for
8:      for $i \leftarrow 1$ to $NP$ do
9:          $\pi_i \leftarrow$ Selection($\pi_i, \upsilon_i^{(1)}, \upsilon_i^{(2)}$)
10:     end for
11:     if restart criterion is verified then
12:         Restart the Population
13:     end if
14: end while
```

The population is initialized with NP uniformly random permutations.

For each population individual π_i, a mutant permutation ν_i is generated according to equation (5) and using the procedure described in Section III.

The crossover between the population individual π_i and the mutant ν_i is performed according to the order based crossover OBX [21]. OBX is a popular crossover operator for permutations and has been widely adopted in the context of LOP (see for example [7] and [19]). Since OBX is not commutative with respect to the parents, we have decided to generate both the offsprings, i.e., $\upsilon_i^{(1)} \leftarrow \text{OBX}(\pi_i, \nu_i)$ and $\upsilon_i^{(2)} \leftarrow \text{OBX}(\nu_i, \pi_i)$. Moreover, we have slightly modified its scheme to take into account the DE parameter $CR \in [0, 1]$. Given the parents π_i and ν_i, for every permutation position $1 \leq k \leq n$, with probability CR, $\upsilon_i^{(1)}[k] \leftarrow \pi_i[k]$ and $\upsilon_i^{(2)}[k] \leftarrow \nu_i[k]$. Then, the remaining values of $\upsilon_i^{(1)}$ and $\upsilon_i^{(2)}$ are assigned following their order of appearance in, respectively, ν_i and π_i.

After being generated, the two offsprings are evaluated and compete with the original population individual π_i to enter the next generation population. Therefore, the next generation population individual π_i' is selected according to $\pi_i' \leftarrow \arg\max \left\{ f(\pi_i), f(\upsilon_i^{(1)}), f(\upsilon_i^{(2)}) \right\}$.

A restart mechanism has been introduced in order to completely avoid the stagnation of the population. The restart is triggered when all the population fitnesses are the same. This roughly corresponds to the case in which all the individuals have the same genotype, but it is more efficient to check. The restart procedure tries to diversify the population (exploration) by also maintaining some information acquired in the previous evolution stages (exploitation). Indeed, half population is randomly regenerated, while the remaining individuals are shuffled each one by a random number in $[1, \binom{n}{2}]$ of adjacent swaps[1].

Finally, it is worthwhile to note that DEP behavior, as in its numerical counterpart, depends from three parameters, namely: the population size NP, the scale factor F, and the crossover probability CR. However, in this work, while NP is left free to be set by the user, F and CR are self-adapted using a variant of the popular scheme proposed in [26], where F is allowed to vary in $(0, 2]$ in order to exploit the extended difference introduced in Section III-B.

V. Experiments

The performances of DEP have been evaluated on a large set of widely known benchmark instances selected from [1]. Namely, we have selected the benchmark suites IO, SGB, MB and XLOLIB for a total of 183 LOP instances. Moreover, in the following, XLOLIB has been split in XLOLIB_150 and XLOLIB_250 containing, respectively, the instances of size 150 and 250. As described in [1], all the instances are normalized thus to allow a more easy comparison with the other optimization methods for LOP[2].

Optimal values are known for IO, SGB and MB instances, while the best known solutions of XLOLIB are reported in [20]. Regarding the size, IO and SGB are composed by relatively small instances, while MB and XLOLIB instances are of greater size. Moreover, as highlighted in [1], the instances in IO and XLOLIB are of real-world type, those of SGB are randomly generated, while the MB instances are somehow in the middle between real-world and random instances.

DEP has been run 10 times for each problem instance thus having a total of $10 \times 183 = 1\,830$ executions. The population size parameter NP has been set to 100 after some preliminary experiments. Similarly to [20], the termination criterion has been set to $10\,000 \times n^2$ fitness evaluations except for XLOLIB_250 were a smaller evaluations budget of $200\,000\,000$ has been adopted.

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

$$ARPD = \left(\sum_{i=1}^{10} \frac{(Best - Alg_i) \times 100}{Best} \right) / 10 \qquad (6)$$

where Alg_i is the final fitness value found by the algorithm Alg in its ith run, and $Best$ is the best known value for the problem instance at hand.

The results for the benchmark suites IO, SGB and MB are provided, respectively, in the tables I, II and III. On these benchmarks, the ARPDs of DEP are computed with respect to the optimal values, while the success rate SR indicates the percentage of executions where DEP found the known optimum. Moreover, results in bold denote that the optimum was reached in every DEP execution.

[1]Note that, as described in Section III, $\binom{n}{2}$ is the maximum number of adjacent swaps that separate two distinct permutations.

[2]Actually, the instances and their optima have been downloaded from www.optsicom.es/lolib/ and www.sc.ehu.es/ccwbayes/members/jceberio/LOPRevisited/.

These results clearly indicate the good performances of DEP. On IO, DEP was able to solve 47 instances over 50, 41 of them in every single run. On SGB, 18 instances over 25 were solved and the worse ARPD has the almost negligible value of 0.02%. Good results have been observed also on the greater MB instances. Here, 21 over 30 instances were solved, 13 of them in every single run, and, most remarkably, the worse ARPD obtained is only of about the 0.01%. Averaging over all the instances with known optima, DEP was able to: (i) solve at least in one execution the 82% of the instances, (ii) solve in every execution the 54% of the instances, (iii) obtain an overall ARPD lower than the 0.01%.

Tables IV and V report the experimental results on, respectively, the benchmark class XLOLIB_150 and XLOLIB_250. Here, the DEP results are compared with the state-of-the-art ARPDs provided in [20]. For every specific instance, the mark in the second column indicates that the best ARPD comes from the algorithm MA_R ($*$) or ILS_R ($\bullet$). Furthermore, figure 1 provides the convergence graph of a typical DEP execution.

Considering that the proposed DEP scheme for LOP is still a preliminary implementation, Tables IV and V show that, although we were not able to match the performances of the state-of-the-art algorithms MA_R and ILS_R, DEP results are anyway satisfactory. Indeed, on XLOLIB_150, the overall ARPD of DEP is 0.66% and it is less than 0.5% greater than that of MA_R, i.e., the best known algorithm to date. The difference is greater on XLOLIB_250, but still small, i.e., around the 1%.

Finally, as shown by the convergence graph in figure 1, DEP typically reaches a good enough solution very soon and employs more than 3/4 of the evolution for small refinements (note that, to favor the readability, the graph has been "truncated" on its x-axis). This aspect clearly reveals that there is room for improvement and that a critical point of the proposed approach resides in its restart mechanism.

Summarizing, the proposed DEP algorithm: (i) was able to easily solve the vast majority of IO, SGB and MB instances, (ii) is competitive with the state-of-the-art algorithms to date on the hardest LOP instances, and (iii) looks to have potentialities for further improvements.

TABLE I: Experimental results on IO benchmarks

Instance	n	ARPD	SR	Instance	n	ARPD	SR
be75eec	50	**0**	**1**	t70k11xx	44	**0**	**1**
be75np	50	0	0	t70l11xx	44	**0**	**1**
be75oi	50	**0**	**1**	t70n11xx	44	**0**	**1**
be75tot	50	**0**	**1**	t70u11xx	44	**0**	**1**
stabu70	60	**0**	**1**	t70w11xx	44	**0**	**1**
stabu74	60	0.02	0.4	t70x11xx	44	**0**	**1**
stabu75	60	0.16	0	t74d11xx	44	**0**	**1**
t59b11xx	44	**0**	**1**	t75d11xx	44	**0**	**1**
t59d11xx	44	**0**	**1**	t75e11xx	44	**0**	**1**
t59f11xx	44	**0**	**1**	t75i11xx	44	**0**	**1**
t59i11xx	44	**0**	**1**	t75k11xx	44	**0**	**1**
t59n11xx	44	**0**	**1**	t75n11xx	44	**0**	**1**
t65b11xx	44	<0.01	0.6	t75u11xx	44	0.02	0.8
t65d11xx	44	**0**	**1**	tiw56n54	56	**0**	**1**
t65f11xx	44	0.10	0.1	tiw56n58	56	**0**	**1**
t65i11xx	44	**0**	**1**	tiw56n62	56	**0**	**1**
t65l11xx	44	**0**	**1**	tiw56n66	56	**0**	**1**
t65n11xx	44	**0**	**1**	tiw56n67	56	**0**	**1**
t65w11xx	44	**0**	**1**	tiw56n72	56	<0.01	0.9
t69r11xx	44	**0**	**1**	tiw56r54	56	**0**	**1**
t70b11xx	44	**0**	**1**	tiw56r58	56	**0**	**1**
t70d11xx	44	**0**	**1**	tiw56r66	56	**0**	**1**
t70d11xxb	44	0.01	0.6	tiw56r67	56	**0**	**1**
t70f11xx	44	**0**	**1**	tiw56r72	56	**0**	**1**
t70i11xx	44	**0**	**1**	usa79	79	0.02	0
Avg ARPD = 0.006				Avg SR = 0.89			

TABLE II: Experimental results on SGB benchmarks

Instance	n	ARPD	SR	Instance	n	ARPD	SR
sgb75.01	75	0.02	0	sgb75.14	75	0.01	0.1
sgb75.02	75	<0.01	0	sgb75.15	75	<0.01	0.1
sgb75.03	75	**0**	**1**	sgb75.16	75	<0.01	0.1
sgb75.04	75	<0.01	0	sgb75.17	75	<0.01	0.3
sgb75.05	75	**0**	**1**	sgb75.18	75	<0.01	0.7
sgb75.06	75	<0.01	0	sgb75.19	75	**0**	**1**
sgb75.07	75	<0.01	0.2	sgb75.20	75	<0.01	0.8
sgb75.08	75	<0.01	0.3	sgb75.21	75	<0.01	0.9
sgb75.09	75	<0.01	0.7	sgb75.22	75	<0.01	0.8
sgb75.10	75	<0.01	0.2	sgb75.23	75	0.01	0
sgb75.11	75	<0.01	0	sgb75.24	75	0.01	0
sgb75.12	75	<0.01	0.8	sgb75.25	75	<0.01	0.7
sgb75.13	75	<0.01	0.1				
Avg ARPD = 0.002				Avg SR = 0.39			

VI. Conclusions and Future Works

In this paper we have described a discrete Differential Evolution algorithm for LOP. The key idea is the use of the algebraic-based differential mutation operator for permutation spaces already defined in [9] for the permutation flowshop scheduling problem (PFSP). This choice is due to the very good performances observed on PFSP. Moreover, both PFSP and LOP are problems defined on the permutation domain, so the scheme can be easily adapted to LOP. A further novelty of this work resides in the possibility to compute, not only truncated, but also extended differences between permutations, thus to allow to use also a scale factor parameter greater than 1 for the discrete differential mutation operator. The DEP algorithm is completed with the Order Based Crossover and a restart scheme. Experiments were conducted on a large and widely known benchmark suite and the results show that the proposed approach is competitive with the state-of-art algorithms for LOP.

As a future line of research, we would like to investigate other selection and restart schemes with the aim of avoiding the population stagnation and the slowdown of the convergence speed observed on the experiments conducted. Another point to be addressed is to try other mutation schemes, for instance based on the generating sets given by all the transpositions (T) and all the insertions (I) by defining randomized sorting algorithms which are able to find a shortest decomposition of a permutation using T and I.

References

[1] Martí, R., Reinelt, G. The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization. Appl. Math. Sciences 175 (2010)

[2] Martí, R., Reinelt, G., Duarte, A. A benchmark library and a comparison of heuristic methods for the linear ordering problem. Computational Optimization and Applications, Vol. 51, Issue 3, pp. 1297–1317 (2012)

[3] Leontief, W. Input-Output Economics. Cambridge University Press (2008)

[4] Charon, I., Hudry, O. A survey on the linear ordering problem for weighted or unweighted tournaments. 4OR, Vol. 5 (1), pp. 5–60 (2007)

TABLE III: Experimental results on MB benchmarks

Instance	n	ARPD	SR	Instance	n	ARPD	SR
r100a2	100	**0**	**1**	r200a0	200	<0.01	0.4
r100b2	100	0.01	0	r200a1	200	<0.01	0.8
r100c2	100	0.01	0	r200b0	200	<0.01	0
r100d2	100	**0**	**1**	r200b1	200	<0.01	0
r100e2	100	**0**	**1**	r200c0	200	<0.01	0.2
r150a0	150	**0**	**1**	r200c1	200	**0**	**1**
r150a1	150	**0**	**1**	r200d0	200	<0.01	0.2
r150b0	150	**0**	**1**	r200d1	200	<0.01	0
r150b1	150	<0.01	0.9	r200e0	200	**0**	**1**
r150c0	150	**0**	**1**	r200e1	200	<0.01	0.2
r150c1	150	**0**	**1**	r250a0	250	<0.01	0
r150d0	150	**0**	**1**	r250b0	250	**0**	**1**
r150d1	150	<0.01	0	r250c0	250	<0.01	0
r150e0	150	**0**	**1**	r250d0	250	<0.01	0.5
r150e1	150	<0.01	0	r250e0	250	<0.01	0.3

Avg ARPD = 0.0009 Avg SR = 0.55

TABLE IV: Experimental results on XLOLIB_150 benchmarks

Instance	Best ARPD	DEP ARPD	Instance	Best ARPD	DEP ARPD
be75eec_150	0.13 ∗	0.32	t70f11xx_150	0.46 ∗	1.35
be75np_150	0.19 ∗	0.68	t70l11xx_150	0.04 ●	0.81
be75oi_150	0.12 ∗	0.40	t70n11xx_150	0.29 ●	0.85
be75tot_150	0.23 ∗	1.18	t74d11xx_150	0.18 ∗	0.83
stabu1_150	0.15 ∗	0.59	t75d11xx_150	0.19 ∗	0.88
stabu2_150	0.09 ∗	0.43	t75e11xx_150	0.33 ●	0.72
stabu3_150	0.11 ∗	0.46	t75k11xx_150	0.13 ∗	0.30
t59b11xx_150	0.28 ∗	0.49	t75n11xx_150	0.25 ●	1.04
t59d11xx_150	0.09 ∗	0.60	tiw56n54_150	0.14 ∗	0.51
t59f11xx_150	0.22 ∗	0.75	tiw56n58_150	0.16 ∗	0.91
t59n11xx_150	0.11 ∗	0.50	tiw56n62_150	0.18 ∗	0.68
t65b11xx_150	0.18 ●	0.57	tiw56n66_150	0.24 ∗	0.55
t65d11xx_150	0.19 ∗	0.87	tiw56n67_150	0.08 ∗	0.47
t65f11xx_150	0.19 ∗	0.82	tiw56n72_150	0.16 ∗	0.51
t65l11xx_150	0.14 ●	0.49	tiw56r54_150	0.06 ∗	0.52
t65n11xx_150	0.14 ●	0.54	tiw56r58_150	0.15 ∗	0.65
t69r11xx_150	0.24 ∗	0.40	tiw56r66_150	0.27 ∗	0.77
t70b11xx_150	0.24 ●	0.57	tiw56r67_150	0.18 ∗	0.69
t70d11xn_150	0.21 ∗	0.59	tiw56r72_150	0.14 ∗	0.70
t70d11xx_150	0.33 ∗	0.96			

Avg ARPDs: DEP 0.66, MA_R 0.19, ILS_R 0.24, MA 0.19, ILS 0.24

TABLE V: Experimental results on XLOLIB_250 benchmarks

Instance	Best ARPD	DEP ARPD	Instance	Best ARPD	DEP ARPD
be75eec_250	0.19 ●	1.34	t70f11xx_250	0.16 ∗	1.33
be75np_250	0.20 ●	1.18	t70l11xx_250	0.27 ●	1.34
be75oi_250	0.17 ●	1.19	t70n11xx_250	0.23 ∗	1.48
be75tot_250	0.12 ∗	1.14	t74d11xx_250	0.13 ∗	1.35
stabu1_250	0.14 ●	1.33	t75d11xx_250	0.15 ∗	1.04
stabu2_250	0.14 ∗	1.30	t75e11xx_250	0.18 ●	1.07
stabu3_250	0.09 ∗	1.02	t75k11xx_250	0.18 ∗	1.39
t59b11xx_250	0.23 ∗	1.33	t75n11xx_250	0.13 ∗	1.77
t59d11xx_250	0.13 ∗	1.16	tiw56n54_250	0.15 ∗	1.44
t59f11xx_250	0.16 ∗	1.36	tiw56n58_250	0.12 ∗	1.12
t59n11xx_250	0.21 ∗	1.42	tiw56n62_250	0.10 ∗	1.24
t65b11xx_250	0.16 ∗	1.65	tiw56n66_250	0.12 ∗	1.33
t65d11xx_250	0.15 ∗	1.30	tiw56n67_250	0.14 ∗	0.91
t65f11xx_250	0.20 ∗	1.11	tiw56n72_250	0.10 ∗	0.97
t65l11xx_250	0.22 ∗	0.80	tiw56r54_250	0.11 ∗	0.79
t65n11xx_250	0.22 ∗	1.33	tiw56r58_250	0.17 ∗	1.16
t69r11xx_250	0.24 ●	1.32	tiw56r66_250	0.11 ∗	0.97
t70b11xx_250	0.21 ∗	1.61	tiw56r67_250	0.16 ∗	1.17
t70d11xn_250	0.19 ∗	1.09	tiw56r72_250	0.15 ∗	1.18
t70d11xx_250	0.17 ∗	1.29			

Avg ARPDs: DEP 1.23, MA_R 0.17, ILS_R 0.24, MA 0.17, ILS 0.24

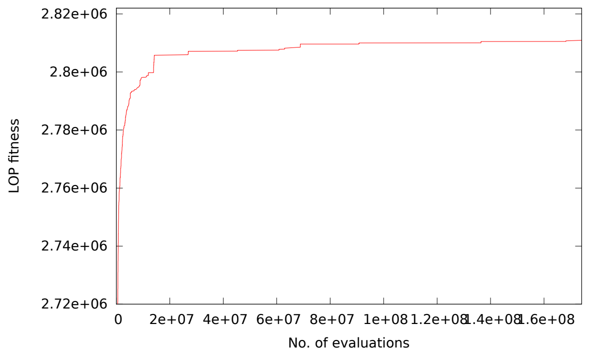

Fig. 1: DEP convergence graph on tiw56r72_150

[5] Glover, F., Klastorin, T., Klingman, D. Optimal weighted ancestry relationships. Management science report series. Univ. of Colorado (1972)

[6] Kemeny, J.G. Mathematics without numbers. Daedalus, Vol. 88, pp. 577–591 (1959)

[7] Schiavinotto, T., Stutzle, T. The linear ordering problem: instances, search space analysis and algorithms. J. Math. Modelling Algorithms, Vol. 3 (4), pp. 367–402 (2004)

[8] Garey, M.R.,Johnson, D.S. Computers and intractability: A guide to the theory of NP-completeness, Freeman (1979)

[9] Santucci, V., Baioletti, M., Milani, A. A Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem with Total Flow Time Criterion. Lecture Notes in Comp. Sc., Vol. 8672, pp. 161–170 (2014)

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

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

[12] Becker, O. Das Helmstadtersche Reihenfolgeproblem – die Effizienz verschiedener Naherungsverfahren. Computer uses in the social sciences, Berichteiner Working Conference, Wien (1967)

[13] Chenery, H.B., Watanabe, T. International comparisons of the structure of production. Econometrica, vol. 26, pp. 487–521 (1958)

[14] Chanas, S., Kobylanski, P. A new heuristic algorithm solving the linear ordering problem. Comp. Opt. and Appl., vol. 6, pp. 191–205 (1996)

[15] Laguna, M., Martí, R., Campos, V. Intensification and diversification with elite tabu search solutions for the linear ordering problem. Computers & Operations Research, Vol. 26 (12), pp. 1217–1230 (1999)

[16] Charon, I., Hudry, O. The noising methods: A generalization of some metaheuristics. European J. of Operat. Res., vol. 135, pp. 86–101 (2001)

[17] Garcia, C.G., Perez-Brito, D., Campos, V., Martí, R. Variable neighborhood search for the linear ordering problem. Computers and Operations Research, vol. 33, pp. 3549–3565 (2006)

[18] Campos, V., Glover, F., Laguna, M., Martí, R. An experimental evaluation of a scatter search for the linear ordering problem. J. of Global Optimization, vol. 21, pp. 397–414 (2001)

[19] Charon, I., Hudry, O. Lamarckian genetic algorithms applied to the aggregation of preferences. Ann. of Op. Res., vol. 80, pp. 281–297 (1998)

[20] Ceberio, J., Mendiburubu A., Lozano, J.A. The linear ordering problem revisited. European J. of Operat. Res., Vol. 241 (3), pp. 686–696 (2015)

[21] Syswerda, G. Schedule optimization using genetic algorithms. In L. Davis, editor, Handbook of Genetic Algorithms. New York (1990)

[22] Kromer, P., Platos, J., and Snasel, V. Differential evolution for the linear ordering problem implemented on CUDA. In Proc. of the 2011 IEEE Con. on Evol. Comp. (CEC) (2011)

[23] Kromer, P., Zelinka, I., and Snasel, V. Can deterministic chaos improve differential evolution for the linear ordering problem? In Proc. of 2014 IEEE Congress on Evol. Comp. (CEC), IEEE, pp. 1443–1448 (2014)

[24] Snasel, V., Kromer, P., Platos, J. Differential Evolution and Genetic Algorithms for the Linear Ordering Problem. Lect. Notes in Comp. Sc., vol. 5711, pp. 139–146 (2009)

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

[26] 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 Ev. Comp. 10 (6), 646–657 (2006)