An Extension of Algebraic Differential Evolution for the Linear Ordering Problem with Cumulative Costs

Chapter · September 2016

DOI: 10.1007/978-3-319-45823-6_12

Marco Baioletti
Università degli Studi di Perugia
59 PUBLICATIONS **245** CITATIONS
SEE PROFILE

Alfredo Milani
Hong Kong Baptist University
94 PUBLICATIONS **354** CITATIONS
SEE PROFILE

Valentino Santucci
Università degli Studi di Perugia
15 PUBLICATIONS **39** CITATIONS
SEE PROFILE

An Extension of Algebraic Differential Evolution for the Linear Ordering Problem with Cumulative Costs

Marco Baioletti, Alfredo Milani, and Valentino Santucci

Department of Mathematics and Computer Science
University of Perugia, Italy
marco.baioletti@unipg.it, alfredo.milani@unipg.it, valentino.santucci@dmi.unipg.it

Abstract. In this paper we propose an extension to the algebraic differential evolution approach for permutation based problems (DEP). Conversely from classical differential evolution, DEP is fully combinatorial and it is extended in two directions: new generating sets based on exchange and insertion moves are considered, and the case $F > 1$ is now allowed for the differential mutation operator. Moreover, also the crossover and selection operators of the original DEP have been modified in order to address the linear ordering problem with cumulative costs (LOPCC). The new DEP schemes are compared with the state-of-the-art LOPCC algorithms using a widely adopted benchmark suite. The experimental results show that DEP reaches competitive performances and, most remarkably, found 21 new best known solutions on the 50 largest LOPCC instances.

Keywords: Algebraic Differential Evolution, Linear Ordering Problem with Cumulative Costs, Permutations neighborhoods

1 Introduction and Related Work

Algebraic Differential Evolution (ADE) [13] is a recently proposed effective metaheuristic for combinatorial optimization. ADE works on discrete search spaces by mimicking the behavior of the numerical Differential Evolution (DE) [16].

In the past, the numerical DE has been applied to combinatorial problems by adopting transformation techniques to decode a numerical vector (genotype) in the corresponding discrete solution (phenotype) in the evaluation step (see for example [2]). However, a single discrete solution can be represented by a potentially infinite number of continuous individuals, thus introducing a one-to-many mapping from the phenotypic to the genotypic space. As a consequence, large plateaus are very likely to be introduced in the search landscape and this is probably the main reason of the poor performances of these combinatorial applications of DE. Conversely, ADE allows to implement a discrete differential mutation operator (the key component of DE) that directly handles the discrete solutions of combinatorial problems.

The only requirement of ADE is that the combinatorial search space at hand must be representable by means of a finitely generated group. This requirement is met in most of the combinatorial search spaces, e.g.: binary strings with the XOR and permutations with the usual composition operator. This algebraic structure cleanly establishes connections with the common solutions neighborhoods usually considered in combinatorial problems. In the case of permutations, the abstract differential mutation has been implemented using a generating set based on the adjacent swap moves [15]. The algebraic Differential Evolution for Permutations (DEP) has been applied to flowshop scheduling problems [13, 12] and to the linear ordering problem [14, 1] where, respectively, state-of-the-art and competitive results have been obtained.

In this paper, we extend our previous works on DEP in four directions: (i) we propose two new implementations of DEP by using generating sets based on exchange and insertion moves [15]; (ii) we extend the definition of the discrete differential mutation by allowing a scale factor parameter larger than 1; (iii) we apply DEP to a popular problem in the field of wireless communications systems, i.e., the Linear Ordering Problem with Cumulative Costs (LOPCC) [3]; (iv) we select other secondary components of DEP in order to tackle LOPCC.

LOPCC has been introduced in [3] as a cumulative variant of the linear ordering problem. Given a complete digraph of n nodes with node weights $d_i \geq 0$ and arc weights $c_{ij} \geq 0$, LOPCC aims to find a permutation of nodes $\pi = \langle \pi_1, \ldots, \pi_n \rangle$ that minimizes

$$f(\pi) = \sum_{i=1}^{n} \alpha_{\pi_i} \tag{1}$$

where the α-costs are backward recursively calculated as

$$\alpha_{\pi_i} = d_{\pi_i} + \textstyle\sum_{j=i+1}^{n} c_{\pi_i \pi_j} \alpha_{\pi_j} \qquad \text{for } i = n, n-1, \ldots, 1. \tag{2}$$

In [3], LOPCC has been proven to be NP-hard. Therefore, the exact algorithms available in literature [3, 11] are effective only when $n \leq 16$. For larger instances, meta-heuristic approaches have been proposed. A Tabu Search (TS) scheme is described in [7]. EvPR is introduced in [8] and mainly consists in a GRASP procedure hybridized with an evolutionary path relinking technique. Finally, [17] proposes the so called Heterogeneous Cellular Processing Algorithm (HetCPA), a pseudo-parallel hybridization of a GRASP procedure with a scatter search scheme. As recently reported in [17], HetCPA and EvPR looks to be the state-of-the-art algorithms so far for LOPCC.

2 Algebraic Differential Evolution for Permutations

As described in [13], the design of the Algebraic Differential Evolution (ADE) mimics that of the classical DE. A population of N candidate solutions $\{x_1, \ldots, x_N\}$ is iteratively evolved by means of the three operators of differential mutation, crossover and selection. Differently from numerical DE, ADE addresses combinatorial optimization problems whose search space is representable by finitely generated groups. Since crossover and selection schemes for combinatorial spaces are widely available in literature, our proposal mainly focuses on the Differential Mutation (DM) operator. DM is widely recognized as the key component of DE [16] and, in its most common variant, generates a mutant v according to

$$v \leftarrow x_{r_0} \oplus F \odot (x_{r_1} \ominus x_{r_2}) \tag{3}$$

where $x_{r_0}, x_{r_1}, x_{r_2}$ are three randomly selected population individuals, while $F > 0$ is the scale factor parameter. In numerical DE, the operators $\oplus, \ominus, \odot$ are the usual vectorial operations of $\mathbb{R}^n$, while, in ADE, their definitions are formally derived using the algebraic structure of the search space.

The triplet $(X, \circ, G)$ is a finitely generated group representing a combinatorial search space if: (i) X is the discrete set of solutions; (ii) $\circ$ is a binary operation on X with the group properties, i.e., closure, associativity, identity (e), and invertibility (x^{-1}); and (iii) $G \subseteq X$ is a finite generating set of the group, i.e., any $x \in X$ has a (not necessarily unique) minimal-length decomposition $\langle g_1, \ldots, g_l \rangle$, with $g_i \in G$, and whose evaluation is x, i.e., $x = g_1 \circ \cdots \circ g_l$. For the sake of clarity, the length of a (minimal) decomposition of x is denoted with $|x|$. Using $(X, \circ, G)$ we can provide the formal definitions of the operators $\oplus, \ominus, \odot$ for ADE. Let $x, y \in X$ and $\langle g_1, \ldots, g_k, \ldots, g_{|x|} \rangle$ be a decomposition of x, then

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

$$x \ominus y := y^{-1} \circ x \tag{5}$$

$$F \odot x := g_1 \circ \cdots \circ g_k \text{ with } k = \lceil F \cdot |x| \rceil \text{ and } F \in [0, 1]. \tag{6}$$

The algebraic structure on the search space naturally defines neighborhood relations among the solutions. Indeed, it induces a colored digraph whose nodes represent the solutions in X and two generic solutions $x, y \in X$ are linked by an arc with color $g \in G$ if and only if $y = x \circ g$. Hence, a one-step search move is directly encoded by a generator, while a composite move can be synthesized as the evaluation of a sequence of generators (a path on the graph). Analogously to what happens in $\mathbb{R}^n$, the elements of X can be dichotomously interpreted both as solutions (nodes on the graph) and as displacements between solutions (path colors on the graph). As detailed in [13], this allows to provide a rational interpretation to the discrete DM of definition (3). The key idea is that the difference $x \ominus y$ is the evaluation of the colors/generators on a shortest path from y to x. This geometric interpretation brings also to some connections with the Geometric DE proposed in [10].

Clearly, the definitions (4) and (5) do not depend on the generating set, thus they are uniquely defined. Conversely, the definition (6) requires a decomposition of x that is not unique in general, therefore a fair stochastic decomposition scheme has been suggested in [13].

The algebraic Differential Evolution for Permutations (DEP) [13] is an implementation of ADE for the search space of permutations. Indeed, permutations of the set $\{1, \ldots, n\}$, together with the usual composition operator, form the widely known symmetric group $\mathcal{S}(n)$, whose neutral element is the identity permutation

e. In the previous series of works [13, 12, 14, 1], the generating set ASW based on adjacent swap moves has been adopted. Formally, $ASW = \{\sigma_i : 1 \leq i < n\}$ where σ_i is the identity permutation with the items i and $i+1$ exchanged. The randomized decomposer for ASW, namely *RandBS*, has been devised by generalizing the classical bubble sort algorithm.

3 Exchange and Insertion based Generating Sets

The generating sets based on exchange and insertion moves are respectively defined as $EXC = \{\epsilon_{ij} : 1 \leq i < j \leq n\}$ and $INS = \{\iota_{ij} : 1 \leq i, j \leq n\}$. ϵ_{ij} is the identity permutation with the items i and j exchanged, while ι_{ij} is the identity where the item i is shifted to position j. Their cardinalities are $|EXC| = \binom{n}{2}$ and $|INS| = (n-1)^2$ and both are proper supersets of ASW. The implementation of DM based on EXC and INS requires a stochastic decomposition algorithm for both the generating sets. Following the same idea used for ASW, we propose the two randomized decomposer for EXC and INS, respectively *RandSS* and *RandIS*.

3.1 RandSS

Any permutation π can be decomposed in a sequence of generators in EXC by sorting π through successive exchange moves. Then, the decomposition is obtained by reversing the sequence of exchanges.

In order to identify the minimal sequence of exchange moves that sorts $\pi \in \mathcal{S}(n)$ we have to consider the cycle representation of π. Indeed, any permutation can be uniquely represented as a product of disjoint cycles [9]. A k-cycle of π is a sequence of k items $(\pi_{i_0}, \ldots, \pi_{i_{k-1}})$ such that, for any $0 \leq j < k$, the item π_{i_j} appears at position $\pi_{i_{(j-1) \bmod k}}$ in π. For example, $\langle 26745831 \rangle = (1268)(37)(4)(5)$. It is important to note that: (i) e is the only permutation with exactly n cycles, and (ii) an exchange of items belonging to the same cycle breaks the cycle into two new cycles, thus increasing the number of cycles by one. Therefore, a minimal decomposition can be obtained by iteratively choosing an exchange move that breaks a cycle.

The randomized decomposer for EXC, namely *RandSS*, is formally defined in Algorithm 1. The cycle weights w_i have been introduced in order to uniformly sample ϵ_{ij} among all the suitable exchanges (lines 7–8). Indeed, any k-length cycle can be broken with $\binom{k}{2} = k(k-1)/2$ different exchanges (line 5). The cycle representation (line 2) can be computed in $\Theta(n)$. The loop at lines 6–11 performs no more than $n-1$ iterations. The operations inside the loop can be performed in $\Theta(n)$. Therefore, the worst-case time complexity of *RandSS* is $\Theta(n^2)$.

Finally, note that *RandSS* generalizes the classical selection sort algorithm. Indeed, it can be shown that selection sort works similarly to *RandSS* but with some limitations: it always breaks the cycle containing the smallest out-of-place item, and it divides the chosen k-length cycle in two cycles of lengths 1 and $k-1$, respectively.

Algorithm 1 RandSS - Randomized Decomposer for EXC

```
1: function RandSS($\pi \in \mathcal{S}(n)$)
2:     $s := \langle\,\rangle$                    ▷ decomposition sequence of $\pi$ incrementally built
3:     $c :=$ getCycles($\pi$)          ▷ $c_i$ is the $i$th cycle of $\pi$; $c_{ij}$ is the $j$th item of cycle $c_i$
4:     for $i := 1, \text{len}(c)$ do
5:         $w_i := \text{len}(c_i)(\text{len}(c_i) - 1)/2$                  ▷ weight of cycle $c_i$
6:     while len($c$) $< n$ do
7:         $c_r :=$ randomly choose a cycle through a roulette wheel basing on the weights $w_i$
8:         $i, j :=$ uniformly choose a pair of indexes from the cycle $c_r$
9:         $\pi := \pi \circ \epsilon_{ij}$
10:        append $\epsilon_{ij}$ to $s$
11:        update the cycles in $c$ and their weights in $w$
12:    reverse the sequence $s$
13:    return $s$
```

3.2 RandIS

The INS decomposition of a permutation $\pi \in \mathcal{S}(n)$ can be obtained by sorting π using only insertion moves. Indeed, the decomposition is the sorting sequence of insertions reversed and inverted, i.e., every ι_{ij} is replaced with its inverse ι_{ji}.[1]

In order to compute the minimal sequence of insertions that sorts π we have to consider the longest increasing subsequence (LIS) of π. A LIS of π is not generally unique and it is defined as one of the longest monotonically increasing subsequence of (not necessarily consecutive) items of π [4]. It is important to note that: (i) e is the only permutation with exactly one LIS of maximal length n, and (ii) an insertion of a new item into a LIS increases the LIS length by one. Therefore, a minimal decomposition can be obtained by iteratively choosing an insertion that moves a new item in a LIS.

The randomized decomposer for INS, namely $RandIS$, is formally defined in Algorithm 2. At line 3 a random LIS L is obtained by modifying the LIS computation algorithm presented in [4][2]. The set U contains the items not in L (line 4). In order to uniformly sample ι_{ij} among all the suitable insertions (lines 9–11), any item in U is weighted by the number of suitable insertions in which it is involved (lines 5–7). The loop at lines 8–14 stops when $len(L) = n$, $U = \emptyset$ and $\pi = e$, therefore no more than $n - 1$ iterations are performed. The operations inside the loop have been implemented in $\Theta(n)$, thus the loop complexity is $\Theta(n^2)$. Moreover, since it is possible to show that the loop complexity dominates the rest, $RandIS$ requires time $\Theta(n^2)$ in the worst-case.

Finally, note that $RandIS$ generalizes the classical insertion sort algorithm. Indeed, classical insertion sort iteratively increases a sorted subsequence maintained at consecutive indexes on the left side of the permutation. Conversely, $RandIS$ allows to spread the sorted subsequence anywhere in the permutation.

4 Extended Differential Mutation

A limit of the discrete differential mutation previously introduced is that the definition of the multiplication operator does not allow to use a scale factor

[1] The inverting step is not considered in Section 3.1 because the exchange generators are self-invertible.

[2] For the sake of space, its description is not reported here.

Algorithm 2 RandIS - Randomized Decomposer for *INS*

```
1:  function RandIS($\pi \in \mathcal{S}(n)$)
2:      $s := \langle\,\rangle$                                ▷ decomposition sequence of $\pi$ incrementally built
3:      $L :=$ get a random LIS of $\pi$
4:      $U := \{1, \dots, n\} \setminus L$                       ▷ set of unassigned items
5:      for all $k \in U$ do
6:          $P^L_{\pi,k} :=$ set of positions in $\pi$ where it is possible to shift item $k$ in order to increase len($L$)
7:          $w_k := |P^L_{\pi,k}|$                             ▷ weight of item $k$
8:      while len($L$) $< n$ do
9:          $r :=$ randomly choose an item in $U$ through a roulette wheel basing on the weights $w_k$
10:         $i :=$ position of $r$ in $\pi$                     ▷ formally, $\pi^{-1}(r)$
11:         $j :=$ uniformly choose a position from $P^L_{\pi,r}$
12:         $\pi := \pi \circ \iota_{ij}$
13:         append $\iota_{ij}$ to $s$
14:         update $L$, $U$ and, for any $k \in U$, update $P^L_{\pi,k}$ and $w_k$
15:     reverse the sequence $s$
16:     invert the generators in $s$
17:     return $s$
```

parameter $F > 1$. Here, the abstract definition (6) is generalized by defining the properties that $z := F \odot x$ with any $F \geq 0$ has to respect, i.e.:

P1 $|z| = \lceil F \cdot |x| \rceil$,
P2 either a decomposition of x is a prefix of a decomposition of z (case $F \in [0, 1]$) or vice versa (case $F > 1$).

Clearly, when $F \in [0, 1]$, definition (6) meets both P1 and P2. For $F > 1$, P1 and P2 mean that, given the decomposition $\langle g_1, \dots, g_{|x|} \rangle$ of x, a possible decomposition for $F \odot x$ is $\langle g_1, \dots, g_{|x|}, g_{|x|+1}, \dots, g_{\lceil F \cdot |x| \rceil} \rangle$ for a suitable choice of the generators $g_{|x|+1}, \dots, g_{\lceil F \cdot |x| \rceil}$. Note that $F \odot x = x \circ g_{|x|+1} \circ \dots \circ g_{\lceil F \cdot |x| \rceil}$. When the search space is finite, its diameter D constrains the maximum value allowed for F to $F^{max}_x = D/|x|$. Anyway, it is possible to extend the definition of $\odot$ by setting $F \odot x := F^{max}_x \odot x$ for any $F > F^{max}_x$. Geometrically, given two generic solutions x, y and $F > 1$, a decomposition of $F \odot (x \ominus y)$ can be interpreted in the search space graph as the sequence of arc colors in a shortest path starting from y, passing for x and extending beyond x. Unfortunately, when $F > 1$ there exist search spaces for which the multiplication operator is not always defined. An example is provided later on.

It is important to observe that, since $\mathcal{S}(n)$ has a finite diameter, the extended case of $F \odot x$ can be implemented by moving x away from e, i.e., towards a diametrically opposite permutation with respect to the identity.

For *ASW*, given $\pi \in \mathcal{S}(n)$, the extended multiplication operator $F \odot \pi$ can be implemented by sorting π in descending order, and composing π with the first $\lceil F \cdot |\pi| \rceil - |\pi|$ adjacent swap generators encountered during the sort. Denoting with r the “reverse” permutation $\langle n, \dots, 1 \rangle$, the sorting step can be performed as $RandBS(r \circ \pi)$, thus reusing the randomized bubble sort proposed in [13]. Therefore, the worst-case complexity is $\Theta(n^2)$ as for $F \in [0, 1]$.

For *EXC*, an algorithm similar to *RandSS* is employed to compute $F \odot \pi$ with $F > 1$. We call it *MergeCycles* and it works by iteratively merging two cycles into one. Indeed, *MergeCycles* iteratively exchanges two items belonging to different cycles in order to merge the two cycles. The iteration stops when

$\lceil F \cdot |\pi| \rceil - |\pi|$ exchanges have been performed. Then, the corresponding exchange generators are composed to the right of π to obtain $F \odot \pi$. Again, the worst-case complexity is $\Theta(n^2)$.

Finally, $\mathcal{S}(n)$ with the INS generating set is an example of a search space where the extended multiplication is not well defined. In order to satisfy the properties P1–P2 above, a necessary condition is that, for all $\pi \in \mathcal{S}(n)$ there must exist at least an insertion $\iota \in INS$ such that $LIS\_length(\pi \circ \iota) = LIS\_length(\pi) - 1$. However, an example can be used to show that this condition is not verified. Indeed, none of the 9 insertions of $\mathcal{S}(4)$ reduces the LIS length of $\langle 2413 \rangle$. Hence, in this paper, we do not consider the case $F > 1$ for the permutations search space generated by insertion moves.

5 Other Algorithmic Components

Though differential mutation is the core operator of DEP, its main scheme requires also a crossover and a selection operator.

In this work we have experimented two popular crossovers for permutation representations, namely, the two point crossover TPII adopted in [13] and the order based crossover OBX used in [1]. Given the parents $\rho', \rho'' \in \mathcal{S}(n)$, both TPII and OBX select a random subset of positions $P \subseteq \{1, \ldots, n\}$ and build the offspring $\upsilon \in \mathcal{S}(n)$ by setting $\upsilon_i \leftarrow \rho'_i$ for any $i \in P$, and inserting the remaining items starting from the leftmost free place of υ and following the order of appearance in ρ''. The difference between TPII and OBX is that TPII uses an interval of positions, while, in OBX, P can be any subset. Furthermore, TPII and OBX have been modified in order to consider the parameter $CR \in [0, 1]$. The modified variants, TPII$^{\text{CR}}$ and OBX$^{\text{CR}}$, constrain the size of P to $|P| = \lceil CR \cdot n \rceil$. Therefore, for each pair of population and mutant individuals x_i, v_i, DEP generates an offspring u_i by applying TPII$^{\text{CR}}$/OBX$^{\text{CR}}$ to v_i and x_i respectively.

Regarding selection, the crowding scheme proposed in [18] is adopted. Each offspring u_j has a closest population individual $closest(u_j)$. Therefore, every population individual x_i is associated to the set of offsprings $U_i = \{u_j : closest(u_j) = x_i\}$. Then, for $1 \leq i \leq N$, the new population individual x'_i is selected to be the fittest among the solutions in $U_i \cup \{x_i\}$. Finally, the computation of the closest population individual has been implemented using the "position based distance" [15] because it is computed in $\Theta(n)$.

6 Experiments

Experiments have been held using the benchmark suites adopted in [7, 8, 17] (and available at http://www.optsicom.es/lopcc): UMTS (100 instances with $n = 16$), LOLIB (42 selected instances with $44 \leq n \leq 60$), RND (three sets of 25 instances of size, respectively, 35, 100 and 150).

DEP population size has been preliminarily set to $N = 80$, while F and CR are self-adapted using the popular jDE scheme [5] modified by introducing

$\hat{F}$, i.e., a cap value for F. In order to test also the extended differential mutation, two choices have been considered for $\hat{F}$, i.e., $\hat{F} \in \{1, 1.2\}$. Hence, every possible combination of $\hat{F}$, generating set and crossover have been tested. Each DEP setting has been run ten times per instance and a run terminates if the best solution so far has not been updated during the last m evaluations[3]. m is set to 5 000, 100 000, 10 000 000 for, respectively, UMTS, LOLIB/RND35 and RND100/RND150 instances.

As in [7, 8, 17], the best result of every DEP setting on every instance has been used to analyze the performances and to compare DEP with the state-of-the-art algorithms HetCPA [17], EvPR [8] and TS [7]. The results of the competitors have been obtained from their respective papers. However, since the full results of EvPR are not available, we have considered optimistic lower bounds for EvPR as done in [17].

Table 1 provides a comparative analysis among the various DEP settings. The average ranks reported for every set of instances show that DEP/INS/1/TPII$^{\mathrm{CR}}$ is the best setting for DEP. Therefore, it has been selected for a further comparison with the state-of-the-art algorithms.

Table 1. Average Ranks among the DEP settings

Bench.	**ASW**	**ASW**	**ASW**	**ASW**	**EXC**	**EXC**	**EXC**	**EXC**	**INS**	**INS**
	$\hat{F}=1$	$\hat{F}=1$	$\hat{F}=1.2$	$\hat{F}=1.2$	$\hat{F}=1$	$\hat{F}=1$	$\hat{F}=1.2$	$\hat{F}=1.2$	$\hat{F}=1$	$\hat{F}=1$
	OBX$^{\mathrm{CR}}$	**TPII$^{\mathrm{CR}}$**	**OBX$^{\mathrm{CR}}$**	**TPII$^{\mathrm{CR}}$**	**OBX$^{\mathrm{CR}}$**	**TPII$^{\mathrm{CR}}$**	**OBX$^{\mathrm{CR}}$**	**TPII$^{\mathrm{CR}}$**	**OBX$^{\mathrm{CR}}$**	**TPII$^{\mathrm{CR}}$**
UMTS	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**
LOLIB	**5.33**	5.55	5.45	6.23	**5.33**	5.44	5.56	5.44	**5.33**	**5.33**
RND35	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**	**5.50**
RND100	5.92	7.04	5.86	5.72	6.62	5.92	4.52	4.80	5.72	**2.88**
RND150	8.32	5.44	9.24	5.76	5.40	2.80	5.04	3.72	7.16	**2.12**
AVG	8.32	5.44	9.24	5.76	5.40	2.80	5.04	3.72	7.16	**2.12**

For each instance set and every algorithm, Table 2, provides both the average rank ($\mathrm{R_{avg}}$) and the average relative percentage deviation (ARPD) from the best solution. By noting that $\mathrm{R_{avg}}$ values are more trustful than the ARPDs because they do not depend on the particular fitness distribution of the instance at hand, Table 2 clearly shows that DEP outperforms the state-of-the-art algorithms on most cases. Moreover, a Friedman+Finner statistical test [6] (with $\alpha = 0.05$) has been conducted. The $\mathrm{R_{avg}}$ values of the algorithms significantly outperformed by DEP are marked with a minus in Table 2. Interestingly, DEP significantly outperforms TS on every instance set except on the small UMTS instances, while the only competitor with similar performances is the hypothetic EvPR. However, since its results are an optimistic assumption, there are good chances that DEP significantly outperforms also EvPR.

Most importantly, DEP found 21 new best known solutions (according to [7, 8, 17]) on the 50 largest instances with $n \in \{100, 150\}$. These new upper bounds

[3] Additionally, a run terminates also if its CPU time exceeds one hour. However, this criterion has been sporadically met only on the RND150 instances.

Table 2. Average ranks and ARPDs among DEP and the state-of-the-art algorithms

Benchmark	**HetCPA**		**Hyp_EvPR**		**TS**		**DEP**	
	R_{avg}	ARPD	R_{avg}	ARPD	R_{avg}	ARPD	R_{avg}	ARPD
UMTS	**2.50**	*0.00*	**2.50**	*0.00*	**2.50**	*0.00*	**2.50**	*0.00*
LOLIB	2.29	*9.00*	**2.24**	14.20	3.24^-	12.59	**2.24**	14.20
RND35	2.44	0.37	**2.22**	*0.00*	3.12^-	0.49	**2.22**	*0.00*
RND100	1.98	2.27	2.10	2.42	4.00^-	15.00	**1.92**	*2.12*
RND150	2.64^-	10.28	1.90	*4.28*	3.66^-	27.13	**1.80**	5.56
OVERALL	2.41	4.38	2.30	*4.18*	3.02^-	11.04	**2.27**	4.38

are provided in Table 3 together with the DEP setting that has obtained the corresponding result. Note that, though DEP/INS/1/TPII^{CR} obtained 10 new best known solutions, also many other DEP settings are present in Table 3. Therefore, another Friedman+Finner test has been conducted by considering a hypothetical DEP^{H} algorithm that produces, for every instance, the best result among all the settings. This test has shown that DEP^{H} would significantly outperform all the competitor algorithms.

Table 3. New best known solutions found by DEP

Instance	**DEP setting**	**Obj.Val.**	**Instance**	**DEP setting**	**Obj.Val.**
t1d100.1	DEP/ASW/1/OBX^{CR}	252.885	t1d100.25	DEP/INS/1/TPII^{CR}	632.586
t1d100.2	DEP/EXC/1.2/TPII^{CR}	286.888	t1d150.2	DEP/EXC/1/TPII^{CR}	163 274.856
t1d100.3	DEP/EXC/1.2/TPII^{CR}	1 288.298	t1d150.6	DEP/INS/1/TPII^{CR}	44 961.697
t1d100.8	DEP/EXC/1.2/OBX^{CR}	2 755.536	t1d150.7	DEP/EXC/1/TPII^{CR}	156 480.244
t1d100.9	DEP/INS/1/TPII^{CR}	61.772	t1d150.10	DEP/INS/1/TPII^{CR}	108 000.853
t1d100.10	DEP/ASW/1.2/TPII^{CR}	155.892	t1d150.12	DEP/INS/1/TPII^{CR}	65 708.550
t1d100.12	DEP/ASW/1.2/OBX^{CR}	231.347	t1d150.13	DEP/INS/1/TPII^{CR}	91 988.932
t1d100.17	DEP/ASW/1.2/TPII^{CR}	715.613	t1d150.16	DEP/INS/1/TPII^{CR}	16 231 674.691
t1d100.20	DEP/ASW/1.2/OBX^{CR}	236.088	t1d150.21	DEP/INS/1/TPII^{CR}	39 663.393
t1d100.22	DEP/ASW/1/OBX^{CR}	144.344	t1d150.22	DEP/INS/1/TPII^{CR}	683 618.275
t1d100.24	DEP/INS/1/TPII^{CR}	464.961			

7 Conclusion and Future Work

The algebraic differential evolution for permutations (DEP) has been extended by introducing the algorithmic implementations for two new generating sets based on exchange and insertion moves, and also by allowing a scale factor parameter larger than one. Moreover, two crossover operators and a crowding selection scheme have been adopted in order to tackle the linear ordering problem with cumulative costs (LOPCC). The proposed approach has been tested on a standard benchmark suite for LOPCC. The experimental results show that DEP reaches state-of-the-art performances by also producing 21 new best known solutions on the 50 largest instances. The notable results obtained by different DEP settings suggest that there is room for further improvement. Therefore, a future line of research will be the design of a meta-DEP scheme that considers all the different settings altogether, for example by using a self-adaptive scheme.

References

1. Baioletti, M., Milani, A., Santucci, V.: Linear ordering optimization with a combinatorial differential evolution. In: 2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp. 2135–2140 (2015)
2. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 6(2), 154–160 (1994)
3. Bertacco, L., Brunetta, L., Fischetti, M.: The linear ordering problem with cumulative costs. European Journal of Operational Research 189(3), 1345–1357 (2008)
4. Bespamyatnikh, S., Segal, M.: Enumerating longest increasing subsequences and patience sorting. Information Processing Letters 76(1–2), 7–11 (2000)
5. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)
6. Derrac, J., Garca, 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 Comp. 1(1), 3–18 (2011)
7. Duarte, A., Laguna, M., Martí, R.: Tabu search for the linear ordering problem withcumulative costs. Computational Optimiz. and Appl. 48(3), 697–715 (2009)
8. Duarte, A., Martí, R., Álvarez, A., Ángel-Bello, F.: Metaheuristics for the linear ordering problem with cumulative costs. Eur. J. of Op. Res. 216(2), 270–277 (2012)
9. Herstein, I.N.: Abstract Algebra, 3rd Edition. Wiley & Sons (1996)
10. Moraglio, A., Togelius, J., Silva, S.: Geometric differential evolution for combinatorial and programs spaces. Evolutionary Computation 21(4), 591–624 (2013)
11. Righini, G.: A branch-and-bound algorithm for the linear ordering problem with cumulative costs. European Journal of Operational Research 186(3), 965–971 (2008)
12. Santucci, V., Baioletti, M., Milani, A.: Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm. AI Communications 29(2), 269–286 (2016)
13. Santucci, V., Baioletti, M., Milani, A.: Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. IEEE Trans. on Evolutionary Computation (to be published – preprint online)
14. Santucci, V., Baioletti, M., Milani, A.: An algebraic differential evolution for the linear ordering problem. In: Proceedings of GECCO 2015. pp. 1479–1480 (2015)
15. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. Computers & Operations Research 34(10), 3143–3153 (2007)
16. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Jour. of Global Opt. 11(4), 341–359
17. Terán-Villanueva, J.D., Fraire Huacuja, H.J., CarpioValadez, J.M., PazosRangel, R., PugaSoberanes, H.J., MartínezFlores, J.A.: A heterogeneous cellular processing algorithm for minimizing the power consumption in wireless communications systems. Computational Optimization and Applications 62(3), 787–814 (2015)
18. Thomsen, R.: Multimodal optimization using crowding-based differential evolution. In: Proceedings of CEC 2004. vol. 2, pp. 1382–1389 (2004)