An Experimental Comparison of Algebraic Differential Evolution using different Generating Sets

Marco Baioletti
University of Perugia
Perugia, Italy
marco.baioletti@unipg.it

Alfredo Milani
University of Perugia
Perugia, Italy
alfredo.milani@unipg.it

Valentino Santucci
University for Foreigners of Perugia
Perugia, Italy
valentino.santucci@unistrapg.it

Umberto Bartoccini
University for Foreigners of Perugia
Perugia, Italy
umberto.bartoccini@unistrapg.it

ABSTRACT

In this paper we provide a comparative empirical analysis of four different generating sets for the algebraic Differential Evolution for Permutations (DEP) applied to the Traveling Salesman Problem (TSP). In particular, DEP has been extended in order to use the reversal moves as generating set. Two different randomized decomposers are proposed for the reversal generators. The experiments have been conducted on a selected set of commonly adopted TSP instances, and the results show the newly proposed generating set leads to better performances with respect to other three generating sets based on alternative search moves.

CCS CONCEPTS

• **Computing methodologies** → **Discrete space search**; **Randomized search**.

KEYWORDS

Algebraic Differential Evolution, Traveling Salesman Problem, Sorting by Reversals

ACM Reference Format:
Marco Baioletti, Alfredo Milani, Valentino Santucci, and Umberto Bartoccini. 2019. An Experimental Comparison of Algebraic Differential Evolution using different Generating Sets. In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3319619.3326854

1 INTRODUCTION

The Algebraic Differential Evolution (ADE) [26] is a recently proposed effective meta-heuristic for combinatorial optimization. ADE works on discrete search spaces by simulating the behavior of the numerical Differential Evolution (DE) [28].

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326854

In the past, the numerical DE has been applied to combinatorial problems by mostly adopting transformation techniques that decode a numerical vector (genotype) into a corresponding discrete solution (phenotype) during the evaluation step (see for example [11]). However, since a single discrete solution can be represented by a potentially infinite number of continuous individuals, the continuous-to-discrete transformation schemes introduce a one-to-many mapping from the phenotypic to the genotypic space. Thereby, large plateaus are very likely to be introduced in the search landscape.

Conversely, ADE allows to implement a discrete differential mutation operator (the key component of DE) that directly handles the discrete solutions of the combinatorial problem at hand.

ADE can be applied to all the optimization problems whose search space can be represented by means of a finitely generated group. This requirement is met by most of the combinatorial search spaces commonly considered like, for instance, the space of binary strings [23] and that of permutations [5]. Furthermore, the algebraic structure underlying these combinatorial space clearly establishes connections with the common solutions neighborhoods usually considered in literature like, for instance, the commonly considered bit-flip neighborhood of the binary space.

In the case of permutations, the algebraic differential mutation has been firstly implemented using a generating set based on the adjacent swap moves [6, 24, 26]. The algebraic Differential Evolution for Permutations (DEP) has been applied to the flowshop scheduling problems [26, 27] and to the linear ordering problem [2, 25] where, respectively, state-of-the-art and competitive results have been obtained.

In [3, 9] DEP has been extended to two generating sets: the one based on, respectively, exchanges and insertion moves. These implementations of DEP obtained new best known solutions on some instances of the linear ordering problem with cumulative costs.

In this paper, we extend DEP by adding a fourth generating set based on the reversal moves. Reversals are one of the key operation in search algorithms for TSP. For instance, they are used in the 2-OPT and 3-OPT algorithms [1].

In order to introduce the reversal moves in our algebraic framework, an algorithm that sorts a permutation with a minimal sequence of reversals is required. However, since sorting-by-reversals

is known to be a NP-hard problem [14], we consider the approximated algorithm proposed by Kecegioglu and Sankoff [19], which finds a sequence of reversal whose length is at most twice the minimum possible length. This algorithm has been randomized in order to produce a random decomposition in terms of reversal generators. Furthermore, we also propose a "more stochastic" randomized decomposer based on a relaxed version of our first proposal.

In this work, we study the behavior of DEP, using the four different generating set, on the Traveling Salesman Problem (TSP) [1], which is a very famous NP-hard problem whose solutions can be represented as permutations. In literature, TSP has been addressed with a variety of techniques, including evolutionary algorithms and other meta-heuristics [18, 20, 22].

Hence, an experimental investigation of the DEP performances on a selected set of widely adopted TSP instances has been conducted in order to understand which are the most suitable generating sets in our scheme. In particular, we are interested to see whether the new generating set – based on the reversal moves – is competitive with respect to the other generating sets. The choice of the TSP problem is thus motivated by the fact that the reversals are essentially equivalent to the 2-OPT moves, i.e., the most "natural" and adopted search moves in the TSP literature [1, 18].

The rest of the paper is organized as follows. Section 2 describes the main concepts of the algebraic differential evolution by also providing the definitions of the generating sets based on adjacent swaps, exchange and insertion moves. Then, the new generating set based on reversal moves is introduced in Section 3, while a comparison of the properties of the four generating sets is provided in Section 4. Next, Section 5 introduces the detailed scheme of the DEP algorithm used in the experimental analysis described in Section 6. Finally, Section 7 concludes the paper by also providing some future lines of research.

2 ALGEBRAIC DIFFERENTIAL EVOLUTION FOR PERMUTATIONS

As described in [26], the design of the Algebraic Differential Evolution (ADE) resembles that of the classical DE. The population $\{x_1, \ldots, x_N\}$ of N candidate solutions 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 spaces form finitely generated groups. Since crossover and selection schemes for combinatorial spaces are widely available in literature, the main focus is on the Differential Mutation (DM) operator. DM is widely recognized as the key component of DE [28] and, in its most common variant, called "rand-1", generates a mutant v according to

$$v \leftarrow x_{r_0} \oplus F \odot (x_{r_1} \ominus x_{r_2}), \tag{1}$$

where $x_{r_0}, x_{r_1}, x_{r_2}$ are three randomly selected population individuals, and $F \in (0, 1]$ is the DE scale factor parameter.

In numerical DE, the operators $\oplus, \ominus, \odot$ are the usual vector operations of $\mathbb{R}^n$, while, in ADE, their definitions are formally derived using the underlying algebraic structure of the search space [10].

The triple $(X, \circ, G)$ is a finitely generated group representing a combinatorial search space if:

- X is the discrete set of solutions in the space;
- $\circ$ is a binary operation on X satisfying the group properties, i.e., closure, associativity, existence of a neutral element (or identity e), and invertibility (x^{-1}); and
- $G \subseteq X$ 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 $g_i \in G$ for all $i \in \{1, \ldots, l\}$, 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 $\oplus, \ominus, \odot$. Let $x, y \in X$ and $\langle g_1, \ldots, g_k, \ldots, g_{|x|} \rangle$ be a minimal decomposition of x, then

$$x \oplus y := x \circ y, \tag{2}$$

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

$$F \odot x := g_1 \circ \cdots \circ g_k, \text{ with } k = \lceil F \cdot |x| \rceil \text{ and } F \in [0, 1]. \tag{4}$$

The algebraic structure on the search space naturally defines neighborhood relations among the solutions. Indeed, it induces a colored digraph whose vertices are 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 $y = x \circ g$. Therefore, a simple one-step move can be 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). In analogy with $\mathbb{R}^n$, the elements of X can be dichotomously interpreted both as solutions (vertices on the graph) and as displacements between solutions (colored paths on the graph). As detailed in [26], this provides a rational interpretation to the discrete DM of definition (1). The key idea is that the difference $x \ominus y$ is the evaluation of the generators in a shortest path from y to x.

Clearly, $\oplus$ and $\ominus$ do not depend on the generating set, thus they are uniquely defined. Conversely, $\odot$ relies on the chosen generating set and, also fixing it, a minimal decomposition is not unique in general. Therefore, $\odot$ is implemented as a stochastic operator, thus requiring a randomized decomposition algorithm for the finitely generated group at hand.

The algebraic Differential Evolution for Permutations (DEP) [26] is an implementation of ADE for the search space of permutations. Indeed, the permutations of the set $[n] = \{1, \ldots, n\}$, together with the usual permutation composition $\circ : [n] \times [n] \to [n]$, form the so-called Symmetric group $\mathcal{S}(n)$. Since $\mathcal{S}(n)$ is finite, it is also finitely generated. Different choices for the generating set are possible and, more interestingly, each generating set can be interpreted as a class of transformations that allow to move from one permutation to another in the search space. In [26] and [3], DEP implementations have been provided for the following three generating sets:

- *ASW*, which is based on *adjacent swap moves* and it is formally defined as

$$ASW = \{\sigma_i : 1 \leq i < n\}, \tag{5}$$

where σ_i is the identity permutation with the items i and $i+1$ exchanged. Hence, given a generic $\pi \in \mathcal{S}_n$, the composition $\pi \circ \sigma_i$ swaps the i-th and $(i+1)$-th items of π. For instance, let $\pi = \langle 3, 5, 2, 4, 1 \rangle$, thus $\pi \circ \sigma_3 = \langle 3, 5, 2, 4, 1 \rangle \circ \langle 1, 2, 4, 3, 5 \rangle = \langle 3, 5, 4, 2, 1 \rangle$.

- *EXC*, which is based on *exchange moves* and it is formally defined as

$$EXC = \{\epsilon_{ij} : 1 \leq i < j \leq n\}, \tag{6}$$

  where ϵ_{ij} is the identity permutation with the items i and j exchanged. Hence, given a generic $\pi \in \mathcal{S}_n$, the composition $\pi \circ \epsilon_{ij}$ swaps the i-th and j-th items of π. For instance, let $\pi = \langle 3, 5, 2, 4, 1\rangle$, thus $\pi \circ \epsilon_{2,5} = \langle 3, 5, 2, 4, 1\rangle \circ \langle 1, 5, 3, 4, 2\rangle = \langle 3, 1, 2, 4, 5\rangle$.
- *INS*, which is based on *insertion moves* and it is formally defined as

$$INS = \{\iota_{ij} : 1 \leq i, j \leq n\}, \tag{7}$$

  where ι_{ij} is the identity permutation where the item i is shifted to position j. Hence, given a generic $\pi \in \mathcal{S}_n$, the composition $\pi \circ \iota_{ij}$ shifts the i-th item in π to position j. For instance, let $\pi = \langle 3, 5, 2, 4, 1\rangle$, thus $\pi \circ \iota_{3,5} = \langle 3, 5, 2, 4, 1\rangle \circ \langle 1, 2, 4, 5, 3\rangle = \langle 3, 5, 4, 1, 2\rangle$.

A decomposition for a generic permutation $\pi \in \mathcal{S}_n$ can be obtained by ordering the items in π using a sorting algorithm that only performs moves from the chosen set, i.e., adjacent swaps for *ASW*, generic exchanges for *EXC*, and insertions for *INS*. The sequence of generators corresponding to the moves performed during the sorting process is annotated, then the sequence is reversed and each generator is replaced with its inverse [7, 8, 26**?**]. Clearly, a sorting algorithm performing an optimal number of moves gives rise to a minimal decomposition.

Optimal randomized decomposers for *ASW*, *EXC*, and *INS* have been implemented by means of generalized and randomized variants of, respectively, the bubble-sort, selection-sort, and insertion-sort algorithms [3, 26]. They have been called *RandBS*, *RandSS*, *RandIS* and each one exploits a different algebraic property of permutations.

Since (the ordered permutation) e is the only permutation with 0 inversions [26], *RandBS* iteratively decreases the number of inversions by swapping two randomly selected adjacent items π_i and π_{i+1} which form an inversion of π, i.e., $\pi_i > \pi_{i+1}$. The time complexity of *RandBS* is $\Theta(n^2)$.

RandSS exploits the fact that e is the only permutation with n cycles (of length 1) in its cycles representation, thus it iteratively increases the number of cycles of π by randomly breaking a randomly selected cycle. Indeed, it turns out that any cycle can be split in two shorter cycles by exchanging any pair of items in the original cycle. The amortized time complexity of *RandSS* is $\Theta(n)$.

RandIS considers that e is the only permutation with a (unique) longest increasing subsequence (LIS) of length n, thus it iteratively extends the LIS length of π by shifting a suitable item in a suitable position. The time complexity of *RandIS* is $\Theta(n^2)$.

For further implementation details, proofs of correctness and complexity we refer the interested reader to [26] and [3].

3 GENERATING SET BASED ON REVERSAL MOVES

Another possible generating set for the permutation group is based on the concept of reversal operation. A reversal $R(i, j)$, with $1 \leq i < j \leq n$, is an operation which, given a permutation $\pi \in \mathcal{S}(n)$, reverses the items in the interval $[i, j]$ of the ordering represented by π. For instance, let $\pi = \langle 4, 9, 5, \underline{3, 8, 2, 1}, 7, 6\rangle$, then the reversal $R(4, 7)$ applied to π produces the permutation $\langle 4, 9, 5, \underline{1, 2, 8, 3}, 7, 6\rangle$. Reversals are widely studied in biology [16].

Here, we provide an algebraic definition of reversal operations and we derive a randomized decomposition algorithm for the corresponding generating set.

Any reversal $R(i, j)$ can be put in correspondence with the permutation $\rho_{ij} = \langle 1, \ldots, i-1, \underline{j, j-1, \ldots, i-1}, i, j+1, \ldots, n\rangle$. Indeed, the application of $R(i, j)$ to a given $\pi \in \mathcal{S}(n)$ is equivalent to $\pi \circ \rho_{ij}$. Since the reversals of the type $\rho_{i,i+1}$ are equivalent to the generators in *ASW*, the set *REV* of all the possible reversals is a generating set of the permutation group. Formally,

$$REV = \{\rho_{ij} : 1 \leq i < j \leq n\}. \tag{8}$$

Furthermore, as for the cases presented in Section 2, a decomposition of a generic permutation, in terms of the generators in *REV*, can be obtained by means of a sorting algorithm that only uses reversal operations.

With this regard, it is easy to see that any permutation can be sorted – thus decomposed – with no more than $n - 1$ reversals[1], however it is not known if this bound is strict or not [15].

In [14], it is proven that the problem of sorting by a minimal number of reversals is NP hard. Hence, also the problem of decomposing a permutation as a minimal sequence of reversals is NP-hard.

Anyway, there exist approximated algorithms which produce a decomposition whose length is at most a small multiple of the minimal decomposition length. Among them, one of the most suitable to be randomized is the greedy algorithm proposed by Kecegioglu and Sankoff in [19], to which we refer with *KS*. The decomposition produced by *KS* has a worst case approximation factor of 2, though, practically the approximation is often very close to the optimal length [19]. Although there are algorithms with theoretically better approximations – for instance, the algorithm of Berman et al. [12] has approximation factor 1.375 – these are much more complex and not suitable for our final goal.

The algorithm *KS* uses the concepts of *breakpoint* and *decreasing strip*. A breakpoint of a permutation π is a position i such that $|\pi(i) - \pi(i + 1)| > 1$, while $\pi(i), \pi(i + 1), \ldots, \pi(j - 1), \pi(j)$ is a decreasing strip if its items are in decreasing order.

If the permutation π is extended in both ends with positions $\pi(0) = 0$ and $\pi(n + 1) = n + 1$, the identity is the only permutation without breakpoints. This is the termination criterion used in *KS* that, at each iteration, reduces the number of breakpoints by selecting a suitable reversal move. Indeed, it is easy to see that a reversal operation $R(i, j)$ can only remove or introduce breakpoints at positions i and j. For the sake of completeness, the pseudo-code of *KS* is provided in Algorithm 1.

Note that, the concept of decreasing strip is used within *KS* to guarantee its termination. As explained in [19], decreasing strips can be quickly verified by maintaining the two arrays "up" and "down". This trick allows to implement *KS* in $\Theta(n^2)$ time.

In the following we introduce two randomized variants of *KS*, namely, *RandRS* and *RandRS2*.

[1] It is enough to iteratively "put in order" the item i by reversing an interval whose two ends are the position i and the position where the item i resides. Clearly, this process requires at most $n - 1$ iterations.

Algorithm 1 Kecegioglu and Sankoff's Greedy algorithm

```
1: function KS(π ∈ S(n))
2:     l := 0
3:     while π contains at least a breakpoint do
4:         l := l + 1
5:         Let ρ_{i_l,j_l} a reversal that removes the most breakpoints
       of π, resolving ties among those that remove one breakpoint
       in favour of reversals that leave a decreasing strip
6:         π := π ∘ ρ_{i_l,j_l}
7:     end while
8:     return ⟨ρ_{i_1,j_1}, ..., ρ_{i_l,j_l}⟩
9: end function
```

3.1 RandRS

We now describe how we have randomized *KS* in order to use it in DEP. By denoting with $nb(\pi)$ the number of breakpoints in the permutation π, our randomization works by iteratively choosing (and applying to π) anyone of the reversals $\rho \in REV$ satisfying one of the following properties in priority order:

(P1) $nb(\pi \circ \rho) = nb(\pi) - 2$;
(P2) $nb(\pi \circ \rho) = nb(\pi) - 1$ and $\pi \circ \rho$ has at least one decreasing strip;
(P3) $nb(\pi \circ \rho) = nb(\pi) - 1$;
(P4) $nb(\pi \circ \rho) = nb(\pi)$.

Property (P4) guarantees that, in the next algorithm iteration, there will be a reversal satisfying one of the properties (P1–P3) [19]. The pseudo-code of the algorithm, called *RandRS*, is provided in Algorithm 2.

Note that, in order to efficiently and randomly select a suitable reversal, any iteration of the main loop applies the reservoir sampling technique (with reservoir size 1) by maintaining the four "reservoir variables" r_1, r_2, r_3, r_4 which contain a uniformly selected random sample from the set of reversals satisfying the properties, respectively, (P1), (P2), (P3) and (P4).

Note that, in line 6, only the reversals removing at least one breakpoint are scanned. This means that the considered reversal removes one breakpoint in one of its ends and, according on what happen in the other end, one of the properties (P1–P4) is necessarily verified.

Then, the reversal in the first non-empty reservoir is selected (lines 29–37), appended to the sorting sequence s (line 38), and applied to π (line 39). Moreover, the breakpoints *BP* are updated accordingly (line 40). Finally, when $BP = \emptyset$, a decomposition is produced by reversing the sorting sequence s (line 42).[2]

RandRS satisfies the same properties of *KS*: it has time complexity $\Theta(n^2)$ and the returned decomposition is longer at most twice the minimal decomposition of the input permutation.

3.2 RandRS2

Although randomized, *RandRS* is anyway a greedy algorithm that, at each iteration, selects a reversal according to the priorities among the rules (P1–P4). It may happen that sometimes only one reversal can be chosen and this aspect may lead to a diversity lost problem when *RandRS* is embedded into an evolutionary algorithm.

For this reason, we have also devised another simpler randomized decomposer, called *RandRS2*, which iteratively chooses a random reversal satisfying anyone of the properties (P1–P4) without any priority among them. The pseudocode of *RandRS2* is provided in Algorithm 3.

In line 5–11, only one reservoir r_1 is used to select, uniformly at random, one of the reversals that removes at least one breakpoint in the current permutation. As aforementioned, a reversal that

[2]Note that, we do not need to also invert the reversals in s, because $\rho^{-1} = \rho$ for any $\rho \in REV$.

Algorithm 2 The algorithm RandRS

```
1: function RandRS(π ∈ S(n))
2:     s ← ⟨ ⟩
3:     BP ← breakpoints of π
4:     while BP ≠ ∅ do
5:         n_1, n_2, n_3, n_4 ← 0
6:         for all ρ_ij removing at least 1 breakpoint do
7:             if ρ_ij satisfies (P1) then
8:                 n_1 ← n_1 + 1
9:                 if random < 1/n_1 then
10:                    r_1 ← ρ_ij
11:                end if
12:            else if ρ_ij satisfies (P2) then
13:                n_2 ← n_2 + 1
14:                if random < 1/n_2 then
15:                    r_2 ← ρ_ij
16:                end if
17:            else if ρ_ij satisfies (P3) then
18:                n_3 ← n_3 + 1
19:                if random < 1/n_3 then
20:                    r_3 ← ρ_ij
21:                end if
22:            else                ▷ ρ_ij automatically satisfies (P4)
23:                n_4 ← n_4 + 1
24:                if random < 1/n_4 then
25:                    r_4 ← ρ_ij
26:                end if
27:            end if
28:        end for
29:        if n_1 > 0 then
30:            ρ ← r_1
31:        else if n_2 > 0 then
32:            ρ ← r_2
33:        else if n_3 > 0 then
34:            ρ ← r_3
35:        else
36:            ρ ← r_4
37:        end if
38:        Append ρ to s
39:        π ← π ∘ ρ
40:        Update BP
41:    end while
42:    Reverse the sequence s
43:    return s
44: end function
```

Algorithm 3 The algorithm RandRS2

```
function RandRS2(π ∈ S(n))
    s ← ⟨ ⟩
    BP ← breakpoints of π
    while BP ≠ ∅ do
        n_1 ← 0
        for all ρ_ij removing at least 1 breakpoint do
            n_1 ← n_1 + 1
            if random < 1/n_1 then
                r_1 ← ρ_ij
            end if
        end for
        Append r_1 to s
        π ← π ∘ r_1
        Update BP
    end while
    Reverse the sequence s
    return s
end function
```

removes a breakpoint – in one of its ends – is guaranteed to satisfy one of the properties (P1–P4). The rest of the algorithm works as in *RandRS*.

Even if there is no theoretical bound for the length of the decomposition produced by *RandRS2*, it has been experimentally observed that the ratio between the length of the sequences produced by RandRS and the sequences produced by RandRS2 is, in average, around 68%.

4 PROPERTIES OF THE GENERATING SETS

Summarizing, DEP can work with four different generating sets: *ASW*, *EXC*, *INS*, and the newly proposed *REV*.

In Table 1, we provide, for each generating set, the number generators, the diameter of the induced Cayley graph, and the time complexity of the corresponding randomized decomposer.

Generating Set	Cardinality	Diameter	Time Complexity
ASW	$n-1$	$\binom{n}{2}$	$\Theta(n^2)$
EXC	$\binom{n}{2}$	$n-1$	$\Theta(n)$
INS	$(n-1)^2$	$n-1$	$\Theta(n^2)$
REV	$\binom{n}{2}$	$\leq n-1$	$\Theta(n^2)$

Table 1: Properties of the generating sets

We now discuss the effect of the classes of search moves, corresponding to every generating set, on a permutation encoding a solution of the (symmetric) Traveling Salesman Problem (TSP).

We first note that adjacent swaps are a special kind of reversals. Indeed, $\sigma_i = \rho_{i,i+1}$ for all $\sigma_i \in ASW$. Hence, without loss of generality, given $\pi \in \mathcal{S}(n-1)$ and $\rho_{ij} \in REV$, the composition $\pi \circ \rho_{ij}$ replaces two arcs of the corresponding TSP tour. In particular, it replaces the arcs $(\pi(i-1), \pi(i))$ and $(\pi(j), \pi(j+1))$ with the arcs $(\pi(i-1), \pi(j))$ and $(\pi(i), \pi(j+1))$.[3] It is interesting to notice that the reversals exactly correspond to the 2-OPT moves widely adopted in the TSP literature [18].

Regarding the generating set *INS*, we consider the effect of the proper insertions, i.e., all the $\iota_{ij} \in INS \setminus ASW$. Given $\pi \in \mathcal{S}(n-1)$, the composition $\pi \circ \iota_{ij}$ replaces three arcs of the TSP tour. In the case $i < j$, it replaces the arcs $(\pi(i-1), \pi(i))$, $(\pi(i), \pi(i+1))$ and $(\pi(j), \pi(j+1))$ with the arcs $(\pi(i-1), \pi(i+1))$, $(\pi(j), \pi(i))$ and $(\pi(j), \pi(i+1))$. The case $i > j$ is analogous. Therefore, proper insertions are a particular case of 3-OPT moves [18].

Finally, the proper exchanges in $EXC \setminus ASW$ have the most disruptive effect since, given $\epsilon_{ij} \in EXC \setminus ASW$ and $\pi \in \mathcal{S}(n-1)$, the composition $\pi \circ \epsilon_{ij}$ replaces four arcs of the TSP tour. The removed arcs are: $(\pi(i-1), \pi(i))$, $(\pi(i), \pi(i+1))$, $(\pi(j-1), \pi(j))$ and $(\pi(j), \pi(j+1))$. The inserted arcs are: $(\pi(i-1), \pi(j))$, $(\pi(j), \pi(i+1))$, $(\pi(j-1), \pi(i))$ and $(\pi(i), \pi(j+1))$. Hence, proper exchanges are a particular case of 4-OPT moves [18].

5 DEP FOR THE TRAVELING SALESMAN PROBLEM

The main scheme of our DEP implementation for the TSP problem is outlined in Figure 4.

Algorithm 4 DEP scheme used in this work

```
Randomly initialize the population {π_1, ..., π_N}
for gen ← 1 to MaxGen do
    for i ← 1 to N do
        v_i ← DifferentialMutation(i, {π_1, ..., π_N}, F)
        v_i ← Crossover(π_i, v_i)
        Evaluate f(v_i)
    end for
    Select the population for the next iteration
    if restart criterion is satisfied then
        Reinitialize the population
    end if
end for
Apply local search to the best individual π_best
```

DEP directly evolves a population $\{\pi_1, \ldots, \pi_N\}$ of N permutations. In this algorithm we use the solution representation [4, 17] which fixes the last city in the TSP tour and uses a permutation of the remaining $n-1$ cities in order to encode a TSP solution[4].

Each permutation uniquely encodes a TSP tour using the solution representation discussed in Section 4.

For every population individual π_i, the differential mutation operator builds a mutant v_i by using the formula (1). The popular self-adaptive scheme jDE [13, 21] is used in order to automatically adapt the value of the parameter $F \in (0, 1]$.

After some preliminary experiments, three alternative crossover operators have been selected: *MPX* (Maximal Preservative CRX), *ER* (Edge Recombination) and *OX1* (Order Crossover). Their descriptions are provided in [20], where it is also claimed that they are among the most used crossovers for the TSP.

[3] For the sake of notation, though $\pi \in \mathcal{S}(n-1)$, we consider $\pi(n) = \pi(0) = n$.

[4] In this way, there is a one-to-one correspondence between permutations in $\mathcal{S}(n-1)$ and tours of n cities.

Regarding selection, the crowding scheme proposed in [29] is adopted in order to slow down population convergence. Each offspring v_j has a closest population individual $closest(v_j)$. Therefore, every population individual π_i is associated to the set of offsprings $U_i = \{v_j : closest(v_j) = \pi_i\}$. Then, for $1 \leq i \leq N$, the new population individual π_i is selected to be the fittest among the solutions in $U_i \cup \{\pi_i\}$.

A restart mechanism is used when all the population individuals converge to the same solution. In that case, $N - 1$ individuals are randomly reinitialized.

At the end of the evolution, a local search operator is applied to the best solution. The local search is based on widely adopted 2-OPT neighborhood (i.e., the reversal moves) and uses the best improvement strategy.

Summarizing, the devised scheme of DEP has three parameters/components to be chosen: the population size N, the generating set used in the differential mutation, and the crossover operator.

6 EXPERIMENTS

An experimental analysis has been conducted on 25 TSP instances selected from the commonly adopted benchmark suite TSPLIB[5]. The size of the instances varies in the range $n \in [14, 100]$. The main goal of this analysis is to investigate the performances of the different generating sets when DEP is applied to the TSP.

Since two different randomized decomposers have been introduced for *REV*, i.e., *RandRS* and *RandRS2*, an additional purpose of the analysis to understand whether the larger amount of randomization in *RandRS2* leads to better results or not. For the sake of notation, we denote by *REV* the results obtained by DEP using the reversal generating set with *RandRS*, and by *REV2* the results obtained using *RandRS2*.

By combining every generating set with the three crossovers *ER*, *MPX* and *OX1*, 15 DEP configurations have been considered.

All the DEP configurations use a population size of 100 individuals and have been executed 20 times per instance, with a 100 000 generations as termination criterion.

The obtained results are shown in Table 4. For each instance i and for each algorithm configuration j, two data are reported. The first is the average relative percentage deviation (ARPD), computed as

$$\frac{1}{20}\sum_{r=1}^{20}\frac{F_{ij}^r - B_i}{B_i} \times 100,$$

where F_{ih}^r is the fitness returned by the algorithm configuration j in its r–th run on the instance i, while B_i is the best fitness value obtained by all the DEP executions for the instance i. Together with the ARPD, we also provide its rank among all the 15 algorithm configurations. ARPD and ranks are then aggregated in the last two rows of the table.

Table 4 clearly shows that the overall best algorithm configurations are *EXC*/*ER*, *REV*/*ER*, *REV2*/*ER*, *REV2*/*MP* and *REV*/*MP*. Although, the algorithm configuration *EXC*/*ER* has been able to produce better solutions for the instances $n = 100$ cities, the algorithm configurations using the reversal moves have anyway similar performances. Moreover, notice that the performances of the *EXC*-based schemes fall down when coupled with the two other crossovers.

In Table 2, we have aggregated the average ARPD and the average ranks for each generating set. It is possible to see that *REV* and *REV2* are the best best generating sets, while *INS* is very close to them and *EXC* and *ASW* are clearly worse, either considering the average ARPD or the average rank. The comparison between *REV* and *REV2* does not produce a clear winner, indeed *REV* has the best ARPD value, while *REV2* reaches the smallest average rank.

	ASW	EXC	INS	REV	REV2
Avg ARPD	7.35	3.97	2.89	2.72	2.80
Avg Rank	11.26	7.65	7.34	6.92	6.83

Table 2: Experimental comparisons of the generating sets

Finally, in Table 3, we provide for each instance the best solutions found by anyone of our algorithm configurations. These results are compared with the best known solution (obtained from the TSPLIB website).It is important to notice that, in many cases, DEP has been able to match the best known solutions, also in some of the larger instances like, for instance, `kroB100` and `kroE100`.

Instance	Best Found	Best Known	Gap
burma14	3330	3323	7
ulysses16	6867	6859	8
gr17	2085	2085	0
gr21	2707	2707	0
ulysses22	7023	7013	10
gr24	1272	1272	0
fri26	937	937	0
bayg29	1610	1610	0
bays29	2020	2020	0
dantzig42	699	699	0
gr48	5046	5046	0
eil51	426	426	0
berlin52	7542	7542	0
brazil58	25395	25395	0
st70	675	675	0
pr76	108159	108159	0
eil76	539	538	1
gr96	55367	55209	158
rat99	1212	1211	1
kroA100	21497	21282	215
kroB100	22141	22141	0
kroC100	20812	20749	63
kroD100	21405	21294	111
kroE100	22068	22068	0
rd100	7911	7910	1

Table 3: Best results by DEP vs Best Known Solution

[5]TSPLIB is available at https://bit.ly/2UBP2Fd.

Instance	ASW/ER	ASW/MP	ASW/O1	EXC/ER	EXC/MP	EXC/O1	INS/ER	INS/MP	INS/O1	REV/ER	REV/MP	REV/O1	REV2/ER	REV2/MP	REV2/O1
burma14	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
ulysses16	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
gr17	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
gr21	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
ulysses22	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
gr24	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
fri26	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8	0 8
bayg29	0.23 15	0.04 12	0 5.5	0.03 11	0 5.5	0 5.5	0.17 14	0 5.5	0 5.5	0 5.5	0 5.5	0 5.5	0.06 13	0 5.5	0 5.5
bays29	0.32 15	0 6.5	0 6.5	0.01 13	0 6.5	0 6.5	0.06 14	0 6.5	0 6.5	0 6.5	0 6.5	0 6.5	0 6.5	0 6.5	0 6.5
dantzig42	0.36 14	0.29 13	0.14 12	0.5 15	0.04 10.5	0.04 10.5	0 5	0 5	0 5	0 5	0 5	0 5	0 5	0 5	0 5
gr48	0.92 11	3.95 15	1.26 14	0.5 4	0.06 1	0.63 7	0.68 8.5	0.61 6	0.92 12	0.71 10	0.51 5	1.07 13	0.46 3	0.39 2	0.68 8.5
eil51	1.24 6	4.41 15	3.83 14	0.92 5	0.85 2.5	3.26 13	0.90 4	2.04 9	2.81 11	0.69 1	1.78 8	2.62 10	0.85 2.5	1.57 7	2.82 12
berlin52	8.74 15	5.61 14	0.66 1	3.08 12	0.87 5	0.67 2	3.5 13	1.38 8	1.03 6	1.42 9	2.18 11	0.77 3	0.79 4	1.81 10	1.18 7
brazil58	2.44 14	4.05 15	2.19 13	0.61 2	0.07 1	0.80 3	0.91 6	1.22 8	1.61 11	0.84 4	1.32 9	1.38 10	1.08 7	1.67 12	0.88 5
st70	7.19 12	13.77 15	9.17 14	2.21 2	2.83 6	7.44 13	2.19 1	2.59 4	4.27 9	2.88 7	2.44 3	5.48 11	2.79 5	2.95 8	4.84 10
eil76	5.75 9	9.21 13	12.66 15	2.54 2	3.46 5	9.30 14	2.88 4	4.55 8	8.62 12	2.81 3	4.25 6	8.10 10	2.44 1	4.51 7	8.21 11
pr76	5.97 12	8.29 15	4.99 9	4.85 8	5.17 10	3.70 4	7.06 13	4.34 7	2.47 1	5.36 11	3.94 6	3.06 3	7.07 14	3.78 5	2.62 2
gr96	3.66 7	10.42 14	10.24 13	3.70 8	5.78 12	14.03 15	1.98 2	4.81 11	4.38 9	2.62 3	2.66 4	4.46 10	1.93 1	3.49 6	3.36 5
rat99	8.35 10	14.28 14	13.53 13	5.25 3	12.12 12	18.40 15	5.72 4	9.23 11	6.36 5	4.48 2	7.20 8	7.57 9	4.33 1	6.99 6	7.17 7
kroA100	18.34 13	23.56 14	25.98 15	3.73 1	10.48 9	13.94 12	4.40 3	4.12 2	11.16 10	5.93 7	4.69 5	8.23 8	4.71 6	4.58 4	12.20 11
kroB100	15.96 12	18.43 13	25.65 15	3.69 1	8.16 8	19.28 14	6.01 7	4.75 3	9.24 10	4.40 2	5.21 5	8.48 9	5.44 6	4.97 4	10.45 11
kroC100	9.96 10	20.63 14	27.30 15	4.88 1	7.39 8	19.93 13	6.14 6	6.35 7	9.82 9	5.92 5	5.21 2	10 11	5.90 4	5.45 3	10.75 12
kroD100	15.81 12	18.60 14	28.79 15	5.28 3	5.91 5	16.10 13	7.79 9	4.99 2	9.5 10	6.31 6	5.44 4	7.76 8	6.82 7	4.64 1	9.85 11
kroE100	17.55 13	20.74 14	23.52 15	4.67 2	13.01 11	16.21 12	4.86 3	3.70 1	11.33 10	6.47 7	4.94 4	9.49 9	6.41 6	5.21 5	8.88 8
rd100	13.10 12	21.10 14	27.98 15	4.72 1	9.19 8	17.53 13	6.5 7	6.04 4	10.84 9	5.39 3	6.28 5	11.63 11	6.29 6	5.25 2	11.07 10
Avg ARPD Avg Rank	5.44 10.72	7.90 12.02	8.72 11.04	2.05 6	3.41 7.28	6.45 9.66	2.47 7.18	2.43 6.56	3.77 8.28	2.25 6.12	2.32 6.32	3.60 8.32	2.30 6.16	2.29 6.20	3.80 8.14

Table 4: ARPDs and Ranks obtained in all the considered TSP instances

7 CONCLUSION AND FUTURE WORKS

In this paper we have studied the performances of DEP, equipped with four different generating sets, on the Traveling Salesman Problem. In particular, we have analyzed the three already proposed generating sets *ASW*, *EXC* and *INS* and the newly introduced generating set *REV* which is based on the widely adopted reversal moves. With this regard, two implemntations of the *REV* randomized decomposer have been considered.

The experimental analysis shows that the reversal moves lead to better results, whichever crossover operator is used.

As a future work, we intend to study the impact of the generating sets in other permutation-based optimization problems. Furthermore, we are also interested in making a similar analysis with other discrete algorithm based on the algebraic framework like, for instance, the algebraic particle swarm optimization [4].

ACKNOWLEDGMENTS

The research described in this work has been partially supported by: the research grant "Fondi per i progetti di ricerca scientifica di Ateneo 2019" of the University for Foreigners of Perugia under the project "Algoritmi evolutivi per problemi di ottimizzazione e modelli di apprendimento automatico con applicazioni al Natural Language Processing"; and by RCB-2015 Project "Algoritmi Randomizzati per l'Ottimizzazione e la Navigazione di Reti Semantiche" and RCB-2015 Project "Algoritmi evolutivi per problemi di ottimizzazione combinatorica" of Department of Mathematics and Computer Science of University of Perugia.

REFERENCES

[1] D. L. Applegate, R. M. Bixby, V. Chvátal, and W. J. Cook. 2006. *The Traveling Salesman Problem: A Computational Study.* Princeton University Press.

[2] M. Baioletti, A. Milani, and V. Santucci. 2015. Linear Ordering Optimization with a Combinatorial Differential Evolution. In *2015 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. 2135–2140. https://doi.org/10.1109/SMC.2015.373

[3] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2016. An Extension of Algebraic Differential Evolution for the Linear Ordering Problem with Cumulative Costs. In *Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings*. 123–133. https://doi.org/10.1007/978-3-319-45823-6_12

[4] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2017. Algebraic Particle Swarm Optimization for the permutations search space. In *2017 IEEE Congress on Evolutionary Computation, CEC 2017, Donostia, San Sebastián, Spain, June 5-8, 2017*. 1587–1594. https://doi.org/10.1109/CEC.2017.7969492

[5] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2017. A New Precedence-Based Ant Colony Optimization for Permutation Problems. In *Simulated Evolution and Learning*. Springer International Publishing, Cham, 960–971. https://doi.org/10.1007/978-3-319-68759-9_79

[6] M. Baioletti, A. Milani, and V. Santucci. 2018. Algebraic Crossover Operators for Permutations. In *2018 IEEE Congress on Evolutionary Computation (CEC 2018)*. 1–8. https://doi.org/10.1109/CEC.2018.8477867

[7] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2018. Automatic Algebraic Evolutionary Algorithms. In *Proc. of Int. Workshop on Artificial Life and Evolutionary Computation (WIVACE 2017)*. Springer International Publishing, Cham, 271–283. https://doi.org/10.1007/978-3-319-78658-2_20

[8] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2018. Learning Bayesian Networks with Algebraic Differential Evolution. In *Proc. of 15th Int. Conf. on Parallel Problem Solving from Nature – PPSN XV*. Springer International Publishing, Cham, 436–448. https://doi.org/10.1007/978-3-319-99259-4_35

[9] Marco Baioletti, Alfredo Milani, and Valentino Santucci. 2018. MOEA/DEP: An Algebraic Decomposition-Based Evolutionary Algorithm for the Multiobjective Permutation Flowshop Scheduling Problem. In *Proc. of European Conference on Evolutionary Computation in Combinatorial Optimization - EvoCOP 2018*. Springer International Publishing, Cham, 132–145. https://doi.org/10.1007/978-3-319-77449-7_9

[10] Marco Baioletti and Valentino Santucci. 2017. Fitness Landscape Analysis of the Permutation Flowshop Scheduling Problem with Total Flow Time Criterion. In *Computational Science and Its Applications – ICCSA 2017*. Springer International Publishing, Cham, 705–716. https://doi.org/10.1007/978-3-319-62392-4_51

[11] J. C. Bean. 1994. Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing* 6, 2 (1994), 154–160. https://doi.org/10.1287/ijoc.6.2.154

[12] Piotr Berman, Sridhar Hannenhalli, and Marek Karpinski. 2002. 1.375-Approximation Algorithm for Sorting by Reversals. In *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*. 200–210. https://doi.org/10.1007/3-540-45749-6_21

[13] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. 2006. Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. *IEEE Transactions on Evolutionary Computation* 10, 6 (2006), 646–657. https://doi.org/10.1109/TEVC.2006.872133

[14] Alberto Caprara. 1997. Sorting by Reversals is Difficult. In *Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB '97)*. ACM, New York, NY, USA, 75–83. https://doi.org/10.1145/267521.267531

[15] Zanoni Dias and Ulisses Dias. 2015. Sorting by Prefix Reversals and Prefix Transpositions. *Discrete Applied Mathematics* 181 (2015), 78 – 89. https://doi.org/10.1016/j.dam.2014.09.004

[16] Guillaume Fertin, Anthony Labarre, Irena Rusu, Stéphane Vialette, and Eric Tannier. 2009. *Combinatorics of genome rearrangements*. MIT press.

[17] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. 1985. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, Vol. 160. Lawrence Erlbaum, 160–168.

[18] Keld Helsgaun. 2009. General k-opt submoves for the Lin–Kernighan TSP heuristic. *Mathematical Programming Computation* 1, 2-3 (2009), 119–163.

[19] John Kececioglu and David Sankoff. 1995. Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. *Algorithmica* 13, 1-2 (1995), 180.

[20] P. Larrañaga, C. M. H. Kuijpers, R.H. Murga, I. Inza, and S. Dizdarevic. 1999. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Artificial Intelligence Review* 13 (1999), 129–170.

[21] Paolo Mengoni, Alfredo Milani, and Yuanxi Li. 2018. Clustering students interactions in eLearning systems for group elicitation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Vol. 10962 LNCS. 398–413. https://doi.org/10.1007/978-3-319-95168-3_27

[22] Yuichi Nagata and Shigenobu Kobayashi. 2013. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS Journal on Computing* 25, 2 (2013), 346–363.

[23] Valentino Santucci, Marco Baioletti, Gabriele Di Bari, and Alfredo Milani. 2019. A Binary Algebraic Differential Evolution for the MultiDimensional Two-Way Number Partitioning Problem. In *Evolutionary Computation in Combinatorial Optimization*. Springer International Publishing, Cham, 17–32. https://doi.org/10.1007/978-3-030-16711-0_2

[24] Valentino Santucci, Marco Baioletti, and Alfredo Milani. 2014. A Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem with Total Flow Time Criterion. In *Parallel Problem Solving from Nature - PPSN XIII - 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings*. 161–170. https://doi.org/10.1007/978-3-319-10762-2_16

[25] Valentino Santucci, Marco Baioletti, and Alfredo Milani. 2015. An Algebraic Differential Evolution for the Linear Ordering Problem. In *Proceedings of GECCO 2015*. 1479–1480. https://doi.org/10.1145/2739482.2764693

[26] Valentino Santucci, Marco Baioletti, and Alfredo Milani. 2016. Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem With Total Flowtime Criterion. *IEEE Trans. Evolutionary Computation* 20, 5 (2016), 682–694. https://doi.org/10.1109/TEVC.2015.2507785

[27] V. Santucci, M. Baioletti, and A. Milani. 2016. Solving Permutation Flowshop Scheduling Problems with a Discrete Differential Evolution Algorithm. *AI Communications* 29, 2 (2016), 269–286. https://doi.org/10.3233/AIC-150695

[28] Rainer Storn and Kenneth Price. [n. d.]. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Jour. of Global Opt.* 11, 4 ([n. d.]), 341–359. https://doi.org/10.1023/A:1008202821328

[29] R. Thomsen. 2004. Multimodal optimization using crowding-based differential evolution. In *Proceedings of CEC 2004*, Vol. 2. 1382–1389. https://doi.org/10.1109/CEC.2004.1331058