Fundamenta Informaticae XXI (2001) 1001–1028
DOI 10.3233/FI-2016-0000
IOS Press

An Experimental Comparison of Algebraic Crossover Operators for Permutation Problems

Marco Baioletti
Department of Mathematics and Computer Science
University of Perugia, Italy
marco.baioletti@unipg.it

Gabriele Di Bari
Department of Mathematics and Computer Science
University of Florence, Italy
gabriele.dibari@unifi.it

Alfredo Milani
Department of Mathematics and Computer Science
University of Perugia, Italy
alfredo.milani@unipg.it

Valentino Santucci
Department of Humanities and Social Sciences
University for Foreigners of Perugia, Italy
valentino.santucci@unistrapg.it

Abstract. Crossover operators are very important components in Evolutionary Computation. Here we are interested in crossovers for the permutation representation that find applications in combinatorial optimization problems such as the permutation flowshop scheduling and the traveling salesman problem. We introduce three families of permutation crossovers based on algebraic properties of the permutation space. In particular, we exploit the group and lattice structures of the space. A total of 34 new crossovers is provided. Algebraic and semantic properties of the operators are discussed, while their performances are investigated by experimentally comparing them with known permutation crossovers on standard benchmarks from four popular permutation problems. Three different experimental scenarios are considered and the results clearly validate our proposals.

Keywords: Permutation crossovers, Algebraic crossovers, Lattice operators

Address for correspondence: valentino.santucci@unistrapg.it

1. Introduction

Crossover operators [1] are a key concept in many Evolutionary Algorithms (EAs), ranging from Genetic Algorithms and Genetic Programming to Differential Evolution [2].

A crossover operator recombines two existing solutions, called parents, to form one or more new solution(s), called offspring(s). Crossovers are often used in combination with a mutation operator. Ideally, crossover and mutation have different purposes. Indeed, the crossover role is to exploit the genetic materials of the parents by recombining them into the offspring(s), while mutation aims to introduce new genetic material into the EA population.

There exist many families of crossover operators that strongly depend on the solutions representation at hand and, with a lesser extent, on the problem to be solved. For instance, one-point and arithmetic crossovers [1, 3] have natural applications when the solutions are represented by, respectively, binary strings and real-valued vectors, while the edge assembly crossover [4] and the partition crossover [5] are specifically designed for the traveling salesman problem.

Here, we are interested in the crossover operators designed for the permutation representation, i.e., those operators that find application in all the permutation-based optimization problems like, for instance, the linear ordering problem [6, 7] and the permutation flowshop scheduling problem [8].

To the best of our knowledge, no previous study has analyzed the relation of a crossover operator with the rich algebraic structure that underlies the permutation space, therefore this article aims to fill this gap by extending our previous work described in [9], where three families of new algebraic crossover operators for the permutation space were defined and studied. In particular, in this work we add new crossover operators, either by using different generating sets with respect to [9], where only the adjacent swaps were considered, and with respect to the path and the vertex selection strategies. Due to its high computational cost, we also decided to not continue to study the greedy strategy for the path selection, but instead we propose a much cheaper strategy, which works quite well.

The first family is mainly based on the algebraic framework we have introduced in [8, 10–12]. In the previous works this algebraic framework has been mainly employed to define combinatorial variants of the Differential Evolution and Particle Swarm Optimization algorithms in order to reach competitive results on some permutation problems. Here we exploit the fact the permutations set forms a finitely generated group, with respect to three different generating sets, by defining a set of 24 group-based algebraic crossover operators.

The algebraic property that permutations space forms a lattice structure – using the generating set based on the adjacent swaps – allows to define two lattice-based algebraic crossover operators.

Furthermore, a third family of 8 hybrid algebraic crossovers is obtained by hybridizing the group-based and lattice-based operators.

Some properties of these new 34 crossovers are discussed, while their performances have been experimentally compared with those of seven popular permutation crossovers from literature.

Three different experimental scenarios have been considered. Namely, the performances of the crossovers are investigated by applying them to several pairs of randomly generated parent permutations, to several pairs of local optima, and embedding them inside a standard genetic algorithm scheme.

Experiments have been run on 12 selected instances from the four most popular permutation prob-

lems, i.e., the linear ordering problem [6], the permutation flowshop scheduling [8], the quadratic assignment problem [13], and the travelling salesman problem [4].

The rest of the papers is organized as follows. Section 2 briefly recalls the algebraic framework for the permutation space. The group-based crossover operator family is introduced in Section 3. Some interesting properties of the crossovers based on adjacent swaps are depicted in Section 4. The lattice-based and the hybrid algebraic crossover families are introduced in 5. The seven permutation crossover operators from literature used in our experimentation are briefly described in Section 6. The experimental analysis is provided in Section 7, while conclusions are drawn in Section 8, where future research directions are also proposed.

2. Algebraic Background

In many combinatorial optimization problems, like for instance the Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP), the purpose is to optimize an objective function f defined on permutation objects. Hence, in these problems, the candidate solutions are permutations of items.

In the following, we briefly recall some concepts that provide an algebraic characterization of the permutation search space [8]. These concepts are used later on to introduce the algebraic crossover operators.

2.1. Permutations and Finitely generated groups

A permutation of the set $[n] = \{1, 2, \ldots, n\}$ is a bijective function $x : [n] \to [n]$. The set of all the permutations of $[n]$ is denoted by $\mathcal{S}_n$. Permutations can be composed by means of the composition operator $\circ$. Given $x, y \in \mathcal{S}_n$, $z = x \circ y$ is defined as $z(i) = x(y(i)),$ for all $i \in [n]$. The set $\mathcal{S}_n$ forms a group with respect to $\circ$, called *symmetric group* . Its neutral element is the identity permutation e, defined as $e(i) = i,$ for all $i \in [n]$. The inverse of $x \in \mathcal{S}_n$ is the permutation x^{-1} defined as $x^{-1}(i) = j$ if and only if $x(j) = i$.

The group $\mathcal{S}_n$ is finitely generated, i.e., there exists a subset H, called *generating set*, of elements of $\mathcal{S}_n$, called *generators*, such that any permutation of $x \in \mathcal{S}_n$ can be expressed as a product

$$x = h_1 \circ \cdots \circ h_l$$

of finitely many generators $h_1, \ldots, h_l \in H$. Actually, $\mathcal{S}_n$ has multiple generating sets. These different choices can be interpreted as a class of transformations that allow to move from one permutation to another in the search space.

In this paper we use three generating sets, which correspond to the most frequently used neighborhoods in search algorithms:

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

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

  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

$$EXC = \{\epsilon_{ij} : 1 \leq i < j \leq n\}, \tag{2}$$

  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

$$INS = \{\iota_{ij} : 1 \leq i, j \leq n\}, \tag{3}$$

  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 $\pi \circ \iota_{3,5} = \langle 3, 5, 2, 4, 1\rangle \circ \langle 1, 2, 4, 5, 3\rangle = \langle 3, 5, 4, 1, 2\rangle$.

Given a generating set H, any permutation may have multiple decompositions in terms of the elements of H. Hence, it is meaningful to restrict the attention to minimal-length decompositions. Although even minimal-length decompositions are not unique in general, for any $x \in \mathcal{S}_n$, it is possible to define the weight $|x|$ as the length of any minimal decomposition of x.

2.2. Cayley graphs

An important concept for any finitely generated group is its Cayley graph. Given a generating set H, the Cayley graph $\mathcal{CG}_H$ for $\mathcal{S}_n$ is the labeled digraph whose vertexes are the elements of $\mathcal{S}_n$ and there is an arc from x to y labeled by $h \in H$ if and only if $y = x \circ h$.

The graph $\mathcal{CG}_H$ is strongly connected, regular, and vertex transitive. These properties guarantee that, for any finite sequence of generators $s \in H^\star$ and for any element $x \in \mathcal{S}_n$, $\mathcal{CG}_H$ has exactly one path which starts from the vertex x and whose arcs are labeled according to s.

In $\mathcal{CG}_H$, for all vertexes $x \in \mathcal{S}_n$, each directed path from the group identity e to x corresponds to a decomposition of x, i.e., if the arc labels occurring in the path are $(h_{j_1}, h_{j_2}, \ldots, h_{j_L})$, then $x = h_{j_1} \circ h_{j_2} \circ \cdots \circ h_{j_L}$. Hence, all the shortest paths from e to x correspond to minimal decompositions of x, thus the length L of these shortest paths is equal to $|x|$.

Furthermore, a generic shortest path from $x \in \mathcal{S}_n$ to $y \in \mathcal{S}_n$ corresponds to a minimal decomposition of $x^{-1} \circ y$. Indeed, if the generators on the path are $(h_{j_1}, h_{j_2}, \ldots, h_{j_L})$, then $y = x \circ (h_{j_1} \circ h_{j_2} \circ \cdots \circ h_{j_L})$, hence $h_{j_1} \circ h_{j_2} \circ \cdots \circ h_{j_L} = x^{-1} \circ y$.

Given $x, y \in \mathcal{S}_n$, we denote by $SP_{x,y}$ the set of all the shortest paths (expressed in terms of sequences of vertexes) from x to y.

Therefore, it is possible to define a metric distance d on $\mathcal{S}_n$. For all $x, y \in \mathcal{S}_n$, $d(x, y)$ is the length of any shortest path in $SP_{x,y}$, or equivalently, $d(x, y) = |x^{-1} \circ y|$. The distance d is known in literature as Kendall's τ distance when $H = ASW$, Cayley distance when $H = EXC$, and Ulam distance when $H = INS$ [14].

Finally, we write $x \sqsubseteq y$ if, for each minimal decomposition $s_x \in H^*$ of x, there exists a minimal decomposition $s_y \in H^*$ of y such that s_x is a prefix of s_y. It is easy to see that $x \sqsubseteq y$ if and only if there exists at least one shortest path in $SP_{e,y}$ which contains x. This is a partial order relation because it may happen that neither $x \sqsubseteq y$ nor $y \sqsubseteq x$. In these cases, x and y are said to be incomparable.

2.3. Vector operations

In a previous series of papers [8, 10, 15–18] we have defined three operations $\oplus, \ominus, \odot$ for a generic finitely generated group. Here we recall their definitions and some of the basic properties for the group $\mathcal{S}_n$.

The definition of these operations lies on the important observation that each element $x \in \mathcal{S}_n$ can be seen as a point-like object, because x is a vertex of $\mathcal{CG}_H$, and also as a vector-like object, because x corresponds to any shortest path from e to x, i.e., to a finite sequence of generators.

The addition $z = x \oplus y$ is defined as the application of the vector $y \in \mathcal{S}_n$ to the point $x \in \mathcal{S}_n$. The result z is computed by choosing a minimal decomposition $s_y = (h_{j_1}, h_{j_2}, \ldots, h_{j_L})$ of y and by finding the end point of the path which starts from x and whose arc labels are the elements of s_y, i.e., $z = x \circ (h_{j_1} \circ h_{j_2} \circ \cdots \circ h_{j_L})$, which simply reduces to

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

Analogously to the Euclidean space, the difference between two points is a vector. Given $x, y \in \mathcal{S}_n$, the difference $x \ominus y$ produces the sequence of labels $(h_{j_1}, h_{j_2}, \ldots, h_{j_L})$ in a shortest path from y to x. Since $h_{j_1} \circ h_{j_2} \circ \cdots \circ h_{j_L} = y^{-1} \circ x$, we define $\ominus$ as

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

Both $\oplus$ and $\ominus$, like their numerical counterparts, are consistent with each other, i.e., $x \oplus (y \ominus x) = y$ for all $x, y \in \mathcal{S}_n$.

Note that $\oplus$ and $\ominus$ are independent from the choice of the generating set. Moreover, since permutations composition and inversion can be computed in $O(n)$ time steps, also $\oplus$ and $\ominus$ have $O(n)$ complexity.

The multiplication $a \odot x$, where a is a real coefficient in $[0, 1]$, is defined for a vector-like $x \in \mathcal{S}_n$ and produces a vector-like $z \in \mathcal{S}_n$ such that $z \sqsubseteq x$ and $|z| = \lceil a \cdot |x| \rceil$. The result of $a \odot x$ can be computed by taking a random minimal decomposition of x in terms of H, truncating it after $\lceil a \cdot |x| \rceil$ generators, and composing the truncated sequence. Since minimal decompositions are not unique, $\odot$ is a stochastic operator. Differently from $\oplus$ and $\ominus$, the operator $\odot$ depends on the generating set H and, indirectly, on the algorithm used to find a random minimal decomposition.

2.4. Randomized decomposers

Here we provide the descriptions of three randomized decomposers for $\mathcal{S}_n$: $RandBS$ for ASW, $RandSS$ for EXC, and $RandIS$ for INS.

2.4.1. RandBS

The randomized decomposer $RandBS$ is actually a randomized implementation of the classical bubble sort algorithm. Its pseudo-code is presented in Figure 1. $RandBS$ sorts x in increasing order (hence obtaining e) by iteratively choosing a random adjacent swap move from the set of adjacent inversions[1]

[1] An adjacent inversion is a pair $(i, i+1)$ such that $x(i) > x(i+1)$

1: **function** RANDBS($x \in \mathcal{S}_n$)
2: $s \leftarrow \langle\,\rangle$
3: $A \leftarrow \{\sigma_i \in ASW : x(i) > x(i+1)\}$
4: **while** $A \neq \emptyset$ **do**
5: $\sigma \leftarrow$ select an element from A uniformly at random
6: $x \leftarrow x \circ \sigma$
7: $s \leftarrow$ Concatenate($\langle\sigma\rangle, s$)
8: $A \leftarrow$ Update(A, σ) ▷ $O(1)$ complexity
9: **end while**
10: Reverse the sequence s
11: **return** s
12: **end function**

Figure 1. Randomized decomposition algorithms for permutations

A. Then, A is efficiently updated by considering that the adjacent swap encoded by σ_i can only affect the three adjacent inversions $(i-1, i)$, $(i, i+1)$, and $(i+1, i+2)$.

Note that the sorting sequence has to be reversed to obtain the decomposition of x. Since the computational cost of $RandBS$ is $O(n^2)$, so is the cost of computing $\odot$ for $H = ASW$.

2.4.2. RandSS

Any permutation $x \in \mathcal{S}_n$ can be decomposed in a sequence of generators in EXC by sorting x 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 $x \in \mathcal{S}_n$ we have to consider the cycle representation of x. Indeed, any permutation can be uniquely represented as a product of disjoint cycles [19]. A k-cycle of x is a sequence of k items $(x(i_0), \ldots, x(i_{k-1}))$ such that, for any $0 \leq j < k$, the item $x(i_j)$ appears at position $i_{(j-1) \bmod k}$ in x. For example, $\langle 2, 6, 7, 4, 5, 8, 3, 1\rangle = (1, 2, 6, 8)(3, 7)(4)(5)$. 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 Figure 2. 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 $\binom{k}{2} = \frac{k(k-1)}{2}$ different exchanges (line 5). The cycle representation (line 2) can be computed in $\Theta(n)$. The loop at lines 7–13 performs no more than $n-1$ iterations, and by amortized analysis, it can be shown that $RandSS$ has $\Theta(n)$ time complexity.

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.

```
1: function RandSS(x ∈ S_n)
2:     s := ⟨ ⟩                          ▷ decomposition sequence of x incrementally built
3:     c := getCycles(x)                 ▷ c_i is the ith cycle of x; c_ij is the jth item of cycle c_i
4:     for i := 1, len(c) do
5:         w_i := len(c_i)(len(c_i) − 1)/2                          ▷ weight of cycle c_i
6:     end for
7:     while len(c) < n do
8:         c_r := randomly choose a cycle through a roulette wheel basing on the weights w_i
9:         i, j := uniformly choose a pair of indexes from the cycle c_r
10:        x := x ∘ ε_ij
11:        Append ε_ij to s
12:        Update the cycles in c and their weights in w
13:    end while
14:    Reverse the sequence s
15:    return s
16: end function
```

Figure 2. RandSS - Randomized Decomposer for EXC

2.5. RandIS

The INS decomposition of a permutation $x \in \mathcal{S}_n$ can be obtained by sorting x 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}.[2]

In order to compute the minimal sequence of insertions that sorts x we have to consider the longest increasing subsequence (LIS) of x. A LIS of x is not generally unique and it is defined as one of the longest monotonically increasing subsequences of (not necessarily consecutive) items of x [20]. 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 described in Figure 3. At line 3 a random LIS L is obtained by modifying the LIS computation algorithm presented in [20][3]. 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 9–16 stops when $len(L) = n$, $U = \emptyset$ and $x = e$, 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

[2]The inverting step is not considered in $RandBS$ and $RandSS$ because the adjacent swaps and the exchange generators are self-invertible.

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

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.

function RANDIS($x \in \mathcal{S}_n$)
  $s := \langle\rangle$ ▷ decomposition sequence of x incrementally built
  $L :=$ get a random LIS of x
  $U := \{1, \ldots, n\} \setminus L$ ▷ set of unassigned items
  for all $k \in U$ **do**
    $P^L_{x,k} :=$ set of positions in x where it is possible to shift item k in order to increase len(L)
    $w_k := |P^L_{x,k}|$ ▷ weight of item k
  end for
  while len$(L) < n$ **do**
    $r :=$ randomly choose an item in U through a roulette wheel basing on the weights w_k
    $i :=$ position of r in x ▷ formally, $x^{-1}(r)$
    $j :=$ uniformly choose a position from $P^L_{x,r}$
    $x := x \circ \iota_{ij}$
    Append ι_{ij} to s
    Update L, U and, for any $k \in U$, update $P^L_{x,k}$ and w_k
  end while
  Reverse the sequence s
  Invert the generators in s
  return s
end function

Figure 3. RandIS - Randomized Decomposer for INS

3. Group-based Algebraic Crossover Operators

In order to define a class of algebraic crossover operators based on the group properties described in Section 2, we need to define the concept of interval on $\mathcal{S}_n$.

Fixed a generating set, and given two permutations $x, y \in \mathcal{S}_n$, the interval $[x, y]$ can be defined in various equivalent ways:

$$[x, y] = \{z \in \mathcal{S}_n \,:\, \exists p \in SP_{x,y} \text{ s.t. } z \text{ appears in } p\}, \tag{6}$$

$$[x, y] = \{z \in \mathcal{S}_n \,:\, z \ominus x \sqsubseteq y \ominus x\}, \tag{7}$$

$$[x, y] = \{z \in \mathcal{S}_n : d(x, z) + d(z, y) = d(x, y)\}. \tag{8}$$

Therefore, the interval $[x, y]$ is, in some sense, the set of permutations between x and y in the Cayley graph of $\mathcal{S}_n$ induced by the chosen generating set.

A group-based algebraic crossover operator AXG can be abstractly defined as any operator which, given two permutations $x, y \in \mathcal{S}_n$, returns a permutation $z = AXG(x, y)$ such that $z \in [x, y]$. Since the notion of interval can be made concrete using anyone of the considered generating sets, we have three classes of group-based algebraic crossovers: the ones based on ASW, EXC and INS, respectively. Anyway, note that, independently of the chosen generating set, $[x, x] = \{x\}$, then $AXG(x, x) = x$ for any possible implementation of AXG.

In any class, the implementations of AXG strongly depend on the method used to select z from $[x, y]$. Ideally, two extreme methods are: randomly select a solution from $[x, y]$ in a uniform way, and choose the fittest solution in $[x, y]$. Clearly, the former one is the least informed and most explorative method, while the latter is the most informed and exploitative. However, it is possible to show that their implementation requires to enumerate all the solutions in $[x, y]$. Since the cardinality of $[x, y]$ is exponential in $d(x, y)$, these two extreme methods are computationally prohibitive.

Here, we propose a more feasible way to obtain similar results. We define a class of concrete AXG crossovers based on a two-phase process: first, a shortest-path $p \in SP_{x,y}$ is constructed, then a vertex z is selected from p.

3.1. Shortest-path selection strategies

Two strategies are devised for the shortest-path construction phase: the random (R), and the tournament (T) construction. Both strategies are structured in two steps. In the first step, a minimal decomposition $(h_{j_1}, h_{j_2}, \ldots, h_{j_L})$ of $y \ominus x$ is obtained. In the second step, the minimal decomposition is converted to the sequence of vertexes $p = (z_0, z_1, \ldots, z_L)$ such that $z_0 = x$ and $z_k = z_{k-1} \circ h_{j_k}$ for all $1 \leq k \leq L$. It is easy to verify that $z_L = y$.

In the random strategy, the minimal decomposition of $y \ominus x$ is computed by using the randomized decomposer of the corresponding generating set: $RandBS$, $RandSS$, or $RandIS$ for, respectively, ASW, EXC, and INS. In [8], [15] and [21] we have showed that this procedure guarantees the largest degree of randomness without increasing the asymptotic complexity.

In the tournament construction procedure (strategy T) the minimal decomposition of $y \ominus x$ is computed in a more informative way. Starting from x, this strategy at each step selects the best neighbor among two randomly chosen neighbors of the incumbent permutation.

The execution time needed by the evaluation of the two neighbors can be reduced by employing a "delta-evaluation" scheme, depending on the problem at hand.

Hence, we devised three procedures $TournBS$, $TournSS$ and $TournIS$, which implement this strategy for the three generating sets by modifying the corresponding randomized decomposer $RandBS$, $RandSS$, and $RandIS$, respectively.

As an example we describe the procedure $TournBS$ for the ASW generating set. Its pseudo-code is presented in Figure 4.

The procedures $TournSS$ and $TournIS$ are obtained from $RandSS$ and $RandIS$, respectively, in an analogous way.

function TOURNBS($x, y \in \mathcal{S}_n$)
  $z \leftarrow y \ominus x$
  $s \leftarrow \langle\,\rangle$
  $A \leftarrow \{\sigma_i \in ASW : z(i) > z(i+1)\}$
  while $A \neq \emptyset$ **do**
    $\sigma_1, \sigma_2 \leftarrow$ select two elements from A uniformly at random
    if $f(x \circ \sigma_1) < f(x \circ \sigma_2)$ **then**
      $\sigma \leftarrow \sigma_1$
    else
      $\sigma \leftarrow \sigma_2$
    end if
    $x \leftarrow x \circ \sigma$
    $z \leftarrow z \circ \sigma$
    $s \leftarrow$ Concatenate$(\langle\sigma\rangle, s)$
    $A \leftarrow$ Update(A, σ) ▷ $O(1)$ complexity
  end while
  Reverse the sequence s
  return s
end function

Figure 4. Tournament decomposition algorithm for permutations

3.2. Vertex selection strategies

Four strategies are considered for the vertex selection phase: uniformly random (R), based on path truncation (T), choosing the best vertex in the path (B), and selecting a random vertex among the best vertexes in the path (P).

Given the selected shortest path $p = (z_0, z_1, \ldots, z_L)$, the vertex selection strategies are straightforward.

The random strategy R simply chooses a solution from p in a uniformly random way, but excluding the two end-points $z_0 = x$ and $z_L = y$.

The truncation strategy T makes use of a further parameter $\alpha \in [0, 1]$ and selects the $\lceil \alpha \cdot L \rceil$-th solution from p. In this work we consider $\alpha = 0.5$. Therefore, the selected solution is as far as possible from both the end-points of p. Hence, this strategy is likely to slow down the loss of diversity when used inside an evolutionary algorithm.

Both the strategies B and P evaluate all the solutions in p. Then, B selects the best solution, while P randomly selects one solution among the best $\lceil \pi \cdot L \rceil$ solutions in p. Here, the parameter π has been fixed to 0.5.

The rationale of this second strategy is to obtain a mix between exploitation and exploration. Finally note that both B and P exclude the end-points x and y. Moreover, as in the tournament strategy for path selection, the "delta-evaluation" techniques (of the problem at hand) can be adopted in order to speed up the process.

3.3. The 24 group-based crossovers

For any chosen generating set, eight AXG implementations are derived by considering any possible combination of the shortest-path and vertex selection strategies.

Therefore, we have 24 group-based algebraic crossovers denoted by following the scheme $AXG\text{-}\mathcal{H}\text{-}\mathcal{AB}$ where: $\mathcal{H} \in \{ASW, EXC, INS\}$ is the generating set, $\mathcal{A} \in \{R, T\}$ is the shortest-path selection strategy, and $\mathcal{B} \in \{R, T, B, P\}$ is the vertex selection strategy.

Note that, for any generating set H, the crossovers $AXG\text{-}\mathcal{H}\text{-}RR$ and $AXG\text{-}\mathcal{H}\text{-}TB$ are fast approximations of, respectively, the most explorative and exploitative way to choose $z \in [x, y]$.

4. Precedences and the ASW-based crossovers

The $AXG\text{-}ASW$ crossovers have a very interesting interpretation in terms of precedences among the items in $[n]$. Indeed, considering the generating set ASW and given $x, y, z \in \mathcal{S}_n$, $z \in [x, y]$ if and only if

$$P(x) \cap P(y) \subseteq P(z) \subseteq P(x) \cup P(y), \tag{9}$$

where $P(x)$ is the set of all the precedence relations induced by x on $[n]$, i.e., $P(x) = \{(i, j) : x^{-1}(i) < x^{-1}(j)\}$. The key observation that allows to prove Property (9) is that the only way to introduce new precedences in z (with respect to x and y), or to avoid the common precedences of x and y, is to move in a non-shortest path between x and y, thus violating the condition of definition (6).

Therefore, given $x, y \in \mathcal{S}_n$, any $AXG\text{-}ASW$ crossover produces a permutation z such that:

1. z contains all the common precedences in x and y, and
2. all the precedences of z come from x or y (no new precedence is generated).

Hence, we say that a $AXG\text{-}ASW$ crossover is *precedence respectful* (Property 1) and *transmits precedences* (Property 2).

5. Lattice-based Algebraic Crossover Operators

Considering again the ASW generating set, $\mathcal{S}_n$ forms a lattice with respect to the partial order relation $\sqsubseteq$, i.e., it admits the two well defined binary operations of meet and join [22]. Given $x, y \in \mathcal{S}_n$, we denote their meet and join by, respectively, $x \wedge y$ and $x \vee y$.

The meet permutation $z = x \wedge y$ has the following properties: (i) $z \sqsubseteq x$ and $z \sqsubseteq y$, (ii) for all $t \in \mathcal{S}_n$, such that $t \sqsubseteq x$ and $t \sqsubseteq y$, then $t \sqsubseteq z$. Hence, $x \wedge y$ is the "greatest" permutation that is "smaller" than both x and y.

The join permutation $z = x \vee y$ has the following properties: (i) $x \sqsubseteq z$ and $y \sqsubseteq z$, (ii) for all $t \in \mathcal{S}_n$, such that $x \sqsubseteq t$ and $y \sqsubseteq t$, then $z \sqsubseteq t$. Hence, $x \vee y$ is the "smallest" permutation that is "greater" than both x and y.

When ASW is considered as generating set, $|x|$ also corresponds to the number of inversions of x, i.e., the number of ordered pairs (i, j), such that $i < j$ and $x^{-1}(i) > x^{-1}(j)$, for $i, j \in [n]$. Let $I(x)$ denote the set of all the inversions of x, we have that $x \sqsubseteq y$ if and only if $I(x) \subseteq I(y)$ (see

for instance [22]). Hence, the meet $x \wedge y$ is the permutation that contains as much as possible of the inversions in $I(x) \cap I(x)$, while the join $x \vee y$ is the permutation that contains all the inversions in $I(x) \cup I(y)$ and as few as possible of the other inversions. These properties guarantee that, for every pair of permutations, meet and join always exist when ASW is considered as generating set [22].

Conversely, INS and EXC do not form a lattice. In these cases, the join is not defined for every pair of permutations.

For EXC it is enough to consider two different cyclic permutations x and y. Since their weigths are maximal, i.e., $|x| = |y| = n - 1$, there is no permutation z such that $x \sqsubseteq z$ and $y \sqsubseteq z$.

For INS, there is a unique maximal weight permutation, but there exist permutations such that there is no insertion operation that decreases their LIS length. Clearly, these permutations do not have a join.

Therefore, in the remaining of this section we only consider the ASW generating set: we first introduce two crossover operators based on meet and join, then we hybridize these crossovers with the previously proposed group-based crossovers.

5.1. Meet and Join Crossover Operators

Here we propose to employ the meet and join as crossover operators denoted by, respectively, $AXL\text{-}Meet$ and $AXL\text{-}Join$.

The operator $AXL\text{-}Meet$ can be computed by an algorithm similar to $RandBS$. Its pseudo-code is provided in Figure 5. $AXL\text{-}Meet$ initializes z to the identity. Then, at every iteration k, it moves z

```
 1: function AXL-MEET(x ∈ S_n, y ∈ S_n)
 2:     x' ← x^{-1}
 3:     y' ← y^{-1}
 4:     z ← e
 5:     A ← {σ_i ∈ ASW : x'(i) > x'(i+1) and y'(i) > y'(i+1)}
 6:     while A ≠ ∅ do
 7:         σ ← select an element from A
 8:         x' ← x' ∘ σ
 9:         y' ← y' ∘ σ
10:         z ← z ∘ σ
11:         A ← Update2(A, σ)                    ▷ done in O(1) complexity
12:     end while
13:     return z
14: end function
```

Figure 5. Meet operator for permutations

one step closer to $x \wedge y$ by composing it with an arbitrary generator that appears at position k in both a minimal decomposition of x and a minimal decomposition of y. When no more common generators are found, i.e., $A = \emptyset$, $z = x \wedge y$. Inversions at lines 2–3 are a simple trick due to the correspondence between a minimal decomposition of x and the reversed sequence of generators that sorts x^{-1}. The procedure $Update2$ updates the set A in constant time, as done in $Update$ (see Subsection 2.4.1).

By denoting with x^R the reverse of x, i.e., $x^R(i) = x(n+1-i)$ for all $i \in [n]$, we can express the "De Morgan"-like property $(x \vee y)^R = x^R \wedge y^R$ which in turn allows to implement $AXL\text{-}Join$ by means of the following equivalence

$$x \vee y = \left(x^R \wedge y^R\right)^R, \tag{10}$$

and thus reusing the $AXL\text{-}Meet$ algorithm.

It is worth to note that $AXL\text{-}Meet(x, y)$ and $AXL\text{-}Join(x, y)$ cannot generate a new individual when $x \sqsubseteq y$ (or $y \sqsubseteq x$). Indeed, in these cases, $x \wedge y = x$ and $x \vee y = y$ (or vice versa).

However, by observing that $I(x \wedge y) \subseteq I(x) \cap I(y)$ and $I(x \vee y) \supseteq I(x) \cup I(y)$ (see for instance [22]), it is possible to show that

$$[x, y] \subseteq [x \wedge y, x \vee y], \tag{11}$$

where the equivalence holds if and only if x and y are comparable. Hence, meet and join, conversely from the group-based crossovers, can generate an offspring with new precedences with respect to x and y.

5.2. Hybrid Algebraic Crossover Operators

In this subsection we introduce the family of crossovers AXH which hybridizes the group-based family AXG for the ASW generating set (see Section 3) with the lattice-based operators (see Section 5).

This hybrid family is motivated from property (11). Given $x, y \in \mathcal{S}_n$, we define $AXH(x, y)$ as a generic crossover that returns an offspring $z \in \mathcal{S}_n$ such that $z \in [x \wedge y, x \vee y]$. The aim is to further explore the interval $[x \wedge y, x \vee y]$ in order to introduce a more variegate set of new precedences (with respect to x and y) in the offspring z.

By exploiting the same shortest path and vertex selection strategies introduced in Section 3, we define eight variants of hybrid algebraic crossovers as follows:

$$\begin{aligned}
AXH\text{-}RR(x, y) &= AXG\text{-}ASW\text{-}RR(x \wedge y, x \vee y),\\
AXH\text{-}RT(x, y) &= AXG\text{-}ASW\text{-}RT(x \wedge y, x \vee y),\\
AXH\text{-}RB(x, y) &= AXG\text{-}ASW\text{-}RB(x \wedge y, x \vee y),\\
AXH\text{-}RP(x, y) &= AXG\text{-}ASW\text{-}RP(x \wedge y, x \vee y),\\
AXH\text{-}TR(x, y) &= AXG\text{-}ASW\text{-}TR(x \wedge y, x \vee y),\\
AXH\text{-}TT(x, y) &= AXG\text{-}ASW\text{-}TT(x \wedge y, x \vee y),\\
AXH\text{-}TB(x, y) &= AXG\text{-}ASW\text{-}TB(x \wedge y, x \vee y),\\
AXH\text{-}TP(x, y) &= AXG\text{-}ASW\text{-}TP(x \wedge y, x \vee y).
\end{aligned}$$

6. Permutation Crossovers in Literature

In this section we review some popular permutation crossover operators available in literature. This review is mainly based on [3, 23, 24].

We denote by x and y the two parent permutations given in input to crossovers, while z indicates the produced offspring. All the operators start from an "empty" sequence z and fill it in an incremental way, until z is a valid permutation.

The Partially-mapped crossover PMX [25] randomly chooses two cut points i_1 and i_2, with $i_1 < i_2$, and copies the portion $y(i_1), \ldots, y(i_2)$ on $z(i_1), \ldots, z(i_2)$. The other positions i of z are filled by copying the elements in the same positions of x. If $x(i)$ is already present in z, the value for $z(i)$ is obtained by applying the mapping $x(j) \leftrightarrow y(j)$ for $j = i_1, \ldots, i_2$, to $x(i)$.

The Order-based crossover $OX1$ [26] randomly chooses two cut points i_1 and $i_2 > i_1$ and copies the portion $x(i_1), \ldots, x(i_2)$ on $z(i_1), \ldots, z(i_2)$. Starting from $i = i_2 + 1$ (or from $i = 1$ if $i_2 = n$), $z(i)$ is filled with $y(j)$, where j is initially such as $j = i$. Each time $y(j)$ is already present in z, j is increased by 1. Once $z(i)$ is copied, i and j are increased by 1. All the increments are clamped in $[n]$, i.e., they produce the result 1, instead of $n + 1$.

Another Order-based crossover, called $OX2$, has been proposed in [27].

A finite sequence $y(i_1), y(i_2), \ldots, y(i_k)$ of random positions of y is chosen and the positions $j_1, \ldots, j_k$ of x are determined such that $x(j_r) = y(i_r)$ for $r = 1, \ldots, k$. Then, $z(i) = x(i)$ for $i \notin \{j_1, \ldots, j_k\}$. The remaining positions $j_1, \ldots, j_k$ of z are filled with the values $y(i_1), y(i_2), \ldots, y(i_k)$ taken in that order.

The cycle crossover CX [28] produces a permutation z such that $z(i) = x(i)$ or $z(i) = y(i)$, for each $i \in [n]$. CX starts by choosing the value for $z(1)$ as $x(1)$ or $y(1)$. If $z(1) = x(1)$, then $z(j) = x(j)$, where j is the position of $y(1)$ in x (i.e. $y(1) = x(j)$). CX continues in this way until the cycle is completed, i.e., when there are no more elements whose position in z is forced. Then, the same process is repeated by starting from the next empty position of z.

The Alternating Position crossover AP [29] fills z by taking the values in an alternating way from x and y (i.e., the first from x, the second from y, and so on) and by omitting the items already present in z.

The Edge Recombination crossover ER [30] has been designed for the TSP problem. For each position $i \in [n]$, the edge set E_i is computed by taking the predecessor and the successor of i in x and y (seen in a circular way). The first position $z(1)$ is filled with a value $k \in \{x(1), y(1)\}$. Then, k is removed from all the set E_i and the value r for $z(2)$ is chosen from E_k by selecting the edge set E_r with the smaller cardinality (ties are randomly broken). The next positions of z are filled in the same way.

The Position-based crossover POS [27] randomly selects some elements $y(i_1), y(i_2), \ldots, y(i_k)$ and copies them in the corresponding positions of z. The remaining positions of z are filled by taking the elements from x in the same order they appear in x, but omitting the elements already present in z.

Finally, note that literature is full of crossover operators purposely defined for certain problems like, for instance in [31], but, to the best of our knowledge, the aforementioned ones are the more general crossovers for permutation problems.

7. Experiments

With the aim to compare the performances of the proposed crossover operators with those described in Section 6, an experimental analysis has been performed on the four most popular permutation-based problems: the linear ordering problem (LOP), the permutation flowshop scheduling problem (PFSP), the quadratic assignment problem (QAP), and the traveling salesman problem (TSP). Three instances per problem have been selected from commonly used benchmark suites as follows:

- LOP: N-be75eec_150, N-stabu1_150, N-t59b11xx;
- PFSP: tai100_5_0, tai100_10_0, tai100_20_0;
- QAP: lipa90a, sko100a, tai100a;
- TSP: kroA100, bier127, pr152.

All the selected instances come from widely adopted benchmark suites: xLOLIB[4] for LOP, Taillard's benchmark suite[5] for PFSP, QAPLIB[6] for QAP, and TSPLIB[7] for TSP.

PFSP has been investigated using the total flowtime as objective criterion [8], while, for TSP, we have adopted the objective function formulation that fixes the last city in the tour [32], thus allowing a one-to-one correspondence between TSP tours and permutations of $n-1$ cities.

Both the parameters α and π, which are present, respectively, on the T vertex selection strategy and on the P path selection strategy, have been set to the value 0.5.

The 34 algebraic crossovers (see Sections 3 and 5) and the 7 competitors (see Section 6) have been compared by means of three different experiments: (i) without embedding them in any algorithm and considering randomly generated parent solutions, (ii) considering local optima solutions, (iii) embedding them in a genetic algorithm. These three experiments are described in, respectively, Subsections 7.1, 7.2, and 7.3.

7.1. Random Experiment

In this experiment, for every problem instance, 5000 pairs of parent permutations have been randomly generated. Then, for each pair of parents, an offspring is generated by each crossover operator. The 41 offsprings are then evaluated and ordered with respect to their fitness values, obtaining a rank value for each crossover in every pair. Then, the rank obtained by every crossover is averaged over all the 5000 pairs. These average ranks, aggregated on every problem for the sake of clarity, are provided in Table 1. The crossovers are ordered according to their overall average rank.

Table 1 clearly shows that almost all the algebraic crossovers which use the vertex selection strategies B and P outperform the competitors. On the other hand, the algebraic crossovers based on the vertex selection R and T strategies achieve lower results.

[4] http://www.optsicom.es/lolib

[5] http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html

[6] http://anjos.mgi.polymtl.ca/qaplib

[7] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

Table 1. Average Ranks in the Random Experiment

Crossover	LOP	PFSP	QAP	TSP	Overall
AXH-RB	* **8.32**	* **4.45**	3.89	* **2.47**	* **4.78**
AXH-TB	8.40	5.05	4.13	2.53	5.03
AXG-ASW-RB	9.44	6.51	* **3.37**	8.21	6.88
AXG-ASW-TB	9.33	6.79	3.46	8.95	7.13
AXG-EXC-RB	8.46	10.28	10.93	15.06	11.18
AXG-EXC-TB	8.80	10.58	10.99	15.13	11.37
AXG-INS-RB	11.13	12.22	15.34	16.31	13.75
AXG-INS-TB	11.01	12.32	15.61	16.33	13.81
AXH-TP	18.92	18.87	19.93	10.30	17.01
AXH-RP	18.86	18.24	19.70	11.29	17.02
AXG-ASW-RP	19.31	18.66	16.91	21.06	18.99
AXG-ASW-TP	19.38	18.83	16.91	21.50	19.16
AXG-EXC-RP	17.80	18.87	18.71	22.39	19.44
AXG-EXC-TP	17.92	19.01	18.75	22.48	19.54
AXG-INS-RP	19.05	19.66	21.02	22.92	20.66
AXG-INS-TP	19.06	19.79	20.96	22.88	20.68
AXL-Meet	20.71	19.72	30.12	12.83	20.84
AXH-TT	26.06	26.37	25.78	13.64	22.96
AXH-TR	25.41	26.72	27.57	13.79	23.37
AXH-RT	26.27	25.38	25.45	17.61	23.68
AXH-RR	25.21	26.65	27.47	16.15	23.87
ER	24.33	25.19	24.83	28.16	25.63
AXL-Join	30.52	30.17	29.35	12.70	25.69
AXG-EXC-TT	24.86	25.39	24.85	28.03	25.78
AXG-EXC-TR	24.99	25.38	24.87	28.00	25.81
AXG-EXC-RT	25.02	25.35	24.92	28.12	25.85
OX1	25.14	25.37	24.88	28.11	25.88
AXG-EXC-RR	24.95	25.56	25.11	28.18	25.95
PMX	25.15	25.49	25.11	28.14	25.97
AXG-INS-RT	25.27	25.56	25.05	28.04	25.98
CX	25.42	25.62	25.03	28.10	26.04
AXG-INS-TT	25.42	25.58	25.00	28.26	26.07
AP	25.64	25.67	25.04	27.94	26.07
AXG-INS-TR	25.39	25.68	24.94	28.33	26.08
AXG-INS-RR	25.48	25.71	24.93	28.26	26.09
AXG-ASW-RR	25.71	25.75	24.99	27.95	26.10
POS	25.61	25.66	25.02	28.22	26.13
AXG-ASW-RT	25.93	25.74	24.95	28.01	26.16
OX2	25.62	25.58	25.11	28.41	26.18
AXG-ASW-TR	25.84	25.84	24.96	28.08	26.18
AXG-ASW-TT	25.85	25.76	25.06	28.12	26.20

Obviously, each crossover which uses the vertex selection strategy B sistematically outperforms the corresponding crossover using P, which in turn outperforms the corresponding crossovers using R and T.

In particular, the two most effective algebraic operators are the hybrid algebraic crossovers coupled with the vertex selection strategy B (i.e., $AXH\text{-}RB$ and $AXH\text{-}TB$), ranking respectively at the first and the second place. Note also they are the best crossover operators in three over four problems In particular, in the TSP instances they reach an outstanding average rank of about 2. Anyway, also in QAP they are among the top four operators.

It is interesting to notice that most of classical crossover operators perform badly on this experiment. Indeed, ER is the best "classical" crossover operator, although it has an overall rank of only 25.63.

Regarding the generating set choice, the class of crossover $AXG - ASW$ appears to outperform the other two classes based on EXC and INS, respectively.

7.2. Local Optima Experiment

The aim of this second experiment is to investigate the performances of the crossover operators when they start from good parent solutions. Hence, in this experiment the parents are local optima.

A collection of 1000 different local optima has been obtained by iteratively performing local searches starting from randomly generated seed solutions. A standard local search scheme has been implemented by considering the widely used insertion neighborhood [14] and the greedy neighbor selection strategy. Then, 5000 pairs of parent solutions are randomly selected from the pool of local optima and the rest of the experiment is conducted as described in Section 7.1. The average ranks are provided in Table 2, where the crossovers are ordered according to their overall rank.

The results shown in Table 2 are somehow different from those observed in the random experiment. From these results, it is possible to determine which are the crossovers that best improve the parent solutions when these are local optima. In this context, the algebraic crossovers $AXG\text{-}*\text{-}*B$, i.e., those combined with the B vertex selection strategy, perform significantly better than the others. In particular, among this subset, the two best crossovers are those based on the adjacent swap generators. Indeed $AXG\text{-}ASW\text{-}RB$ and $AXG\text{-}ASW\text{-}TB$ are the best choice in every problem.

Among the classical crossovers, ER performs well only on TSP, while on average, the best ones are PMX and CX that, anyway, are far away from the best performing algebraic crossover.

It is interesting to notice that most of the hybrid crossovers, which were the best ones in the random experiment, now perform much worse. In particular, the $AXH\text{-}*$ operators are also outperformed by the $AXG\text{-}ASW\text{-}*$ crossovers. Let note that the space explored by the latter is a subset of the space explored by the former, i.e., $[x, y] \subseteq [x \wedge y, x \vee y]$, where both the intervals consider the ASW generating set (see Section 5.2). Therefore, it happens that, since there is no completely greedy path selection strategy (see Section 3.1), the hybrid crossovers may select permutations outside $[x, y]$ which, being an interval between two local optima, is more likely to contain good solutions.

7.3. Genetic Algorithm Experiment

A final experiment has been held by embedding the crossover operators in a Genetic Algorithm (GA).

Table 2. Average Ranks in the Local Optima Experiment

Crossover	LOP	PFSP	QAP	TSP	Overall
AXG-ASW-RB	* **2.54**	* **2.87**	* **2.90**	2.73	* **2.76**
AXG-ASW-TB	* **2.54**	2.97	2.94	* **2.70**	2.79
AXG-EXC-RB	5.28	4.78	3.29	5.34	4.67
AXG-EXC-TB	5.35	4.75	3.36	5.26	4.68
AXG-INS-RB	3.90	4.04	11.17	4.01	5.78
AXG-INS-TB	3.90	4.08	11.36	3.89	5.81
AXG-INS-TP	12.45	10.89	23.52	11.12	14.50
AXG-INS-RP	12.25	10.62	23.44	11.89	14.55
PMX	19.65	15.10	10.73	13.18	14.67
CX	17.48	13.55	7.98	20.37	14.85
AXG-ASW-TP	10.06	15.56	22.18	18.55	16.59
OX2	18.54	15.18	15.68	21.11	17.63
POS	18.54	15.24	15.71	21.06	17.64
AXG-EXC-TP	22.53	19.00	8.83	21.06	17.85
AXG-EXC-RP	22.53	19.02	8.84	21.75	18.03
AXG-ASW-RP	10.22	15.53	22.61	23.98	18.09
AXG-INS-TR	16.64	14.85	27.63	13.89	18.25
AXG-INS-RR	16.53	14.64	27.53	14.64	18.34
OX1	21.20	22.94	21.16	8.44	18.44
AXG-ASW-TR	12.81	20.28	28.54	22.97	21.15
AXG-INS-TT	21.49	19.52	30.29	16.88	22.05
AXG-INS-RT	21.61	19.53	30.21	17.76	22.28
AP	11.91	15.56	30.49	32.09	22.51
AXG-ASW-RR	13.10	20.32	28.85	28.24	22.63
AXG-EXC-TR	26.99	25.63	12.95	27.10	23.17
AXG-EXC-RR	27.37	25.83	13.23	27.88	23.58
AXH-RB	30.01	26.46	16.35	23.15	23.99
AXG-ASW-TT	16.35	24.12	30.46	25.41	24.09
AXH-TB	30.36	27.45	16.55	23.14	24.37
AXG-ASW-RT	16.78	24.17	30.46	31.61	25.76
ER	40.56	39.27	30.41	8.43	29.67
AXG-EXC-TT	32.84	32.90	19.13	34.67	29.89
AXG-EXC-RT	33.78	33.58	19.80	35.78	30.73
AXH-TP	33.88	33.12	27.90	31.00	31.48
AXH-RP	33.58	32.69	27.77	34.58	32.15
AXH-TT	32.36	33.05	31.68	34.31	32.85
AXH-RT	32.10	32.60	31.54	39.00	33.81
AXL-Meet	39.12	38.51	34.27	25.56	34.36
AXH-TR	36.05	35.75	32.74	33.47	34.50
AXL-Join	39.92	39.55	33.79	25.66	34.73
AXH-RR	35.88	35.51	32.73	37.33	35.36

A standard steady-state GA scheme has been considered by taking inspiration from the one used in [23]. A population of N solutions is randomly generated. At every iteration, two parent solutions are randomly selected from the current population. An offspring is generated by means of the crossover operator and it undergoes mutation with probability p_{mut}. The (possibly mutated) offspring is then evaluated and competes with the worst population individual to have a place in the next iteration population.

Different variants of the GA have been implemented by using all the crossover operators considered in this paper. The mutation is performed by applying a random insertion move [14] with probability $p_{mut} = 0.05$. The population size has been set to 50, and each algorithm execution terminates after $1\,000\,000$ iterations.

Moreover, we have designed a meta-crossover operator $AX\text{-}Comb$, that at each iteration, selects one of our 34 operators according to a simple adaptive strategy. For each algebraic crossover, a karma value is maintained. At each iteration, $AX\text{-}Comb$ randomly chooses an operator with probabilities proportional to the karma values. All karmas are initialized to 1. Then, after the selected operator is applied, it receives a karma reward or penalty if the produced offspring, respectively, enters or not the next iteration population. The reward consists of $nit/1000$ points of karma, where nit is the iteration number (in this way, late successes have a greater reward), while the penalty consists in decreasing the karma by 1 point, though karma values are truncated to 1 if smaller.

For each instance and for every crossover operators, 20 executions have been performed. The final fitness values of every execution have been averaged, thus obtaining a ranking of the crossovers in every instance. These average ranks, aggregated on every problem, are provided in Table 3. The crossovers are ordered according to their overall rank.

The results in Table 3 show that the group-based crossovers based on INS perform, in average, better than the other operators. In particular they outperform all the classical operators. Notice that also in this case the hybrid crossovers are not competitive in general. Moreover, the group-based crossovers based on ASW are clearly worse than the operators based on the other generating sets.

Another interesting fact is that there is no clear winner among the vertex selection strategies. Indeed it is possible to find also the strategies R and T near the top of the standing, although they did not perform well on the first two experiments.

Regarding the results on the single problems, in LOP $AXG\text{-}INS\text{-}TT$ is the best operator and it is also among the top 5 on PFSP, where the best one is $AXG\text{-}INS\text{-}TP$. QAP is best solved by operators based on the EXC generating set. Finally, in TSP the best two operators are AP and ER, which however were explicitly designed for this problem. The best algebraic operators are in this case the hybrid $AXH\text{-}RP$ and $AXH\text{-}RR$, both based on the path selection strategy R.

Regarding $AX\text{-}Comb$, it must be noted that it obtains intermediate results among the algebraic crossovers. In fact, it is in average outperformed by the newly introduced INS and EXC based crossovers, except that in the TSP case, where it is outperformed by $AXH\text{-}RP$ and $AXH\text{-}RR$. This intermediate behavior is plausibly due to the warm-up time required by $AX\text{-}Comb$ to select a suitable operator.

We have also studied the diversity of the population during the generations by computing the Kendall's-τ distances between the parent solutions at every iteration. We have chosen as a typical case an execution on the QAP instance tai100a. In Figure 6 we present the most representative behaviours

Table 3. Average Ranks in the GA Experiment

Crossover	LOP	PFSP	QAP	TSP	Overall
AXG-INS-RP	11.67	6.67	10.17	17.33	* **11.46**
AXG-INS-TT	* **9.33**	7.67	16.17	14.67	11.96
AXG-INS-RB	17.33	7.00	14.33	14.33	13.25
AXG-INS-TB	15.67	6.00	15.33	16.33	13.33
AXG-INS-RT	11.00	8.33	12.67	22.33	13.58
AXG-INS-TP	17.67	* **2.67**	18.33	15.67	13.58
AXG-INS-TR	12.67	6.67	18.33	21.00	14.67
AXG-INS-RR	15.33	9.67	23.00	12.67	15.17
AXG-EXC-RT	20.67	19.67	4.67	16.67	15.42
AXG-EXC-RB	20.67	11.00	* **4.33**	26.33	15.58
ER	11.33	14.33	34.00	5.33	16.25
AX-Comb	27.67	10.33	15.33	12.33	16.42
AXG-EXC-RP	17.00	13.00	7.00	31.00	17.00
PMX	17.00	20.33	5.00	29.67	18.00
POS	18.67	22.33	9.33	24.33	18.67
OX1	22.67	13.67	22.00	17.00	18.83
AP	13.67	22.33	37.67	* **5.00**	19.67
CX	19.33	11.00	17.67	30.67	19.67
OX2	20.00	21.67	10.17	29.33	20.29
AXG-EXC-TT	18.67	22.67	7.67	33.00	20.50
AXG-EXC-TB	28.00	17.00	5.67	34.67	21.33
AXG-EXC-TR	20.33	24.00	7.00	35.67	21.75
AXG-ASW-RR	33.33	19.83	10.00	28.33	22.88
AXG-EXC-TP	29.33	25.33	11.83	25.33	22.96
AXG-EXC-RR	15.33	23.67	30.33	23.00	23.08
AXH-RB	17.33	33.17	27.67	16.33	23.63
AXH-RR	17.00	39.33	33.00	8.00	24.33
AXH-TR	19.33	36.00	29.33	14.00	24.67
AXH-RP	26.33	36.00	28.67	7.67	24.67
AXG-ASW-RP	14.67	19.00	28.33	37.33	24.83
AXL-Join	17.67	31.67	41.33	11.33	25.50
AXH-RT	27.00	28.33	28.67	18.67	25.67
AXH-TB	27.33	33.33	26.67	20.00	26.83
AXG-ASW-TP	24.00	23.33	31.33	30.00	27.17
AXH-TP	25.33	38.33	28.33	18.00	27.50
AXH-TT	30.67	35.67	35.33	12.00	28.42
AXG-ASW-TR	27.33	30.67	30.67	25.33	28.50
AXG-ASW-RB	32.33	21.33	30.67	33.00	29.33
AXG-ASW-TT	36.67	37.33	28.67	16.67	29.83
AXG-ASW-TB	25.00	23.00	33.67	40.67	30.58
AXL-Meet	30.67	35.33	41.00	22.67	32.42
AXG-ASW-RT	40.00	34.33	31.67	29.33	33.83

of six class of crossovers: the lattice-based, three group-based (one per generating set), the hybrid ones and the classical operators. It is possible to notice that the lattice based crossover $AXL\text{-}Meet$ maintains a greater level of diversity, followed by, in order of diversity loss, the EXC and INS based AXG crossovers, PMX, $AXH\text{-}RB$ and, lastly, the ASW based operator. Comparing these behaviours with the results of Table 3, it looks that the appropriate level of diversity loss is that observed in the EXC and INS crossovers.

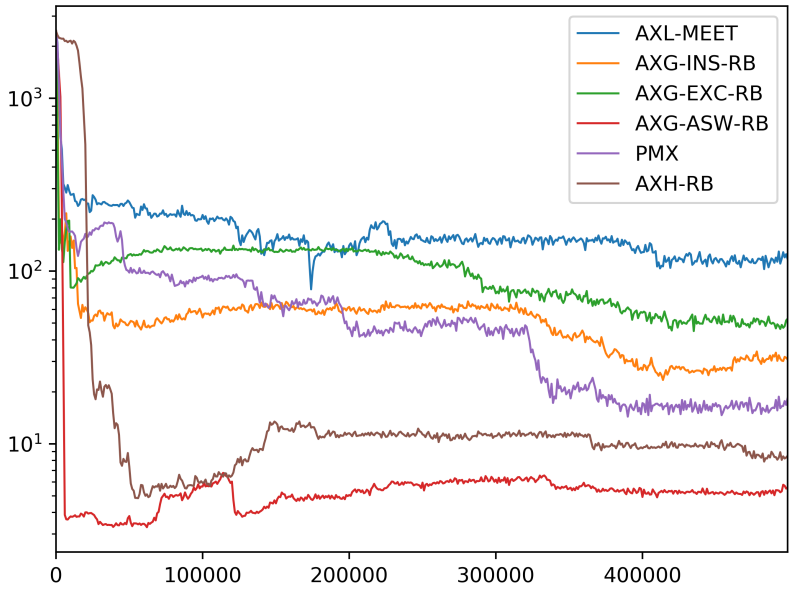

Figure 6. Behaviour of Kendall's-τ distances

7.4. Analysis of the Computational Time

In this section we show the experimental results about the computational times analysis of the proposed crossovers, both as stand-alone operators and embedded in the genetic algorithm.

In Table 4 we report for each crossover operators the average computational time, in seconds, observed in one representative instance of every problem: N-stabu1_150 (for LOP), tai100_10_0 (for PFSP), and bier127 (for TSP). These data are obtained from the experiments described in Section 7.1.

Crossovers are ordered with respect to the average rank obtained on all the instances.

The fastest crossover operators are obviously six of the classical operators PMX, AP, CX, $OX1$, POS, and $OX2$, which all operate in $O(n)$. Similar times are needed by the lattice crossover operators $AXL\text{-}Join$ and $AXL\text{-}Meet$. Although their worst-case complexity is the similar to that of $AXG\text{-}ASW\text{-}RR$, they result to be faster. In fact, we have computed the average number of steps s

Table 4. Computational Times in the Random Experiment (in seconds)

Crossover	LOP	PFSP	QAP	TSP
PMX	$2.0 \cdot 10^{-7}$	$4.0 \cdot 10^{-7}$	$2.0 \cdot 10^{-7}$	$2.0 \cdot 10^{-7}$
AP	$1.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$6.0 \cdot 10^{-7}$	$8.0 \cdot 10^{-7}$
CX	$1.0 \cdot 10^{-6}$	$1.0 \cdot 10^{-6}$	$8.0 \cdot 10^{-7}$	$1.0 \cdot 10^{-6}$
OX1	$1.2 \cdot 10^{-6}$	$1.2 \cdot 10^{-6}$	$8.0 \cdot 10^{-7}$	$1.2 \cdot 10^{-6}$
POS	$2.0 \cdot 10^{-6}$	$2.0 \cdot 10^{-6}$	$1.4 \cdot 10^{-6}$	$1.8 \cdot 10^{-6}$
OX2	$2.2 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$	$1.4 \cdot 10^{-6}$	$1.8 \cdot 10^{-6}$
AXL-Join	$2.4 \cdot 10^{-6}$	$2.4 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$
AXL-Meet	$2.4 \cdot 10^{-6}$	$2.4 \cdot 10^{-6}$	$1.6 \cdot 10^{-6}$	$2.2 \cdot 10^{-6}$
AXG-EXC-RT	$9.2 \cdot 10^{-6}$	$7.8 \cdot 10^{-6}$	$6.0 \cdot 10^{-6}$	$7.6 \cdot 10^{-6}$
AXG-EXC-RR	$9.6 \cdot 10^{-6}$	$9.0 \cdot 10^{-6}$	$6.2 \cdot 10^{-6}$	$7.8 \cdot 10^{-6}$
ER	$2.2 \cdot 10^{-5}$	$2.2 \cdot 10^{-5}$	$1.2 \cdot 10^{-5}$	$1.6 \cdot 10^{-5}$
AXG-ASW-RT	$9.5 \cdot 10^{-5}$	$5.7 \cdot 10^{-5}$	$4.2 \cdot 10^{-5}$	$6.8 \cdot 10^{-5}$
AXG-ASW-RR	$9.6 \cdot 10^{-5}$	$5.8 \cdot 10^{-5}$	$4.2 \cdot 10^{-5}$	$6.8 \cdot 10^{-5}$
AXG-INS-RT	0.0001	$7.2 \cdot 10^{-5}$	$5.1 \cdot 10^{-5}$	$8.1 \cdot 10^{-5}$
AXG-INS-RR	0.0001	$7.6 \cdot 10^{-5}$	$5.2 \cdot 10^{-5}$	$8.2 \cdot 10^{-5}$
AXG-EXC-RP	0.0007	0.0007	0.0001	$1.9 \cdot 10^{-5}$
AXG-EXC-RB	0.0007	0.0009	0.0001	$2.2 \cdot 10^{-5}$
AXH-RT	0.0002	0.0001	$8.2 \cdot 10^{-5}$	0.0001
AXH-RR	0.0002	0.0001	$8.3 \cdot 10^{-5}$	0.0001
AXG-INS-RP	0.0008	0.0009	0.0002	$9.2 \cdot 10^{-5}$
AXG-EXC-TT	0.0015	0.0019	0.0003	$3.7 \cdot 10^{-5}$
AXG-EXC-TR	0.0015	0.0019	0.0003	$3.7 \cdot 10^{-5}$
AXG-INS-RB	0.0008	0.0009	0.0002	$9.4 \cdot 10^{-5}$
AXG-INS-TT	0.0015	0.0017	0.0003	0.0002
AXG-EXC-TP	0.0022	0.0028	0.0004	$4.8 \cdot 10^{-5}$
AXG-INS-TR	0.0015	0.0018	0.0003	0.0002
AXG-EXC-TB	0.0022	0.0028	0.0004	$5.1 \cdot 10^{-5}$
AXG-INS-TP	0.0021	0.0025	0.0004	0.0002
AXG-INS-TB	0.0022	0.0026	0.0004	0.0002
AXG-ASW-RP	0.0284	0.0238	0.0034	0.0004
AXG-ASW-RB	0.0286	0.0238	0.0035	0.0006
AXH-RP	0.0547	0.0420	0.0064	0.0008
AXH-RB	0.0551	0.0419	0.0066	0.0012
AXG-ASW-TT	0.0566	0.0455	0.0067	0.0009
AXG-ASW-TR	0.0566	0.0467	0.0067	0.0009
AXG-ASW-TP	0.0849	0.0659	0.0101	0.0012
AXG-ASW-TB	0.0850	0.0630	0.0102	0.0014
AXH-TT	0.1092	0.0758	0.0128	0.0017
AXH-TR	0.1092	0.0798	0.0128	0.0017
AXH-TP	0.1637	0.1092	0.0191	0.0024
AXH-TB	0.1641	0.1051	0.0193	0.0028

needed by $AXL\text{-}Meet$ with respect to the permutation length n and we have found that the s grows as a linear function of n, with a R^2 score very close to 1.

As expected, ER is the slowest classical operator, because it performs a more complex elaboration.

Among the group based and hybrid crossover families, the results show that the computational times are influenced by the following factors:

- the path selection strategy (R/T),
- the vertex selection strategy (R/T/B/P),
- the generating set,
- whether the crossover is pure or hybrid.

In particular, *ceteris paribus*, the path selection R is faster than T, because it does not require any fitness evaluation. Among the vertex selection strategies, R and T are comparable and much faster than B and P, which require d fitness evaluations (where d is the length of the minimal decomposition, which is equal to the distance between x and y).

The times observed by the crossovers which use the EXC and INS generating sets are, in general, smaller than those using ASW. In fact, in the first two cases $d = O(n)$, while in the last case $d = O(n^2)$. When the path strategy is T or the vertex selection strategy is P or B, the differences among the generating sets are particularly relevant.

The differences among the problems are mainly due to the complexity of the fitness evaluation and the existence of fast delta-evaluation procedures (for instance, these procedure for LOP are described in [33]). In particular, the TSP objective function requires $O(n)$ time, while in the other three problems the cost is $O(n^2)$ and this explains why the computational times for TSP are much smaller.

In Table 5, we report the average computational times observed needed by 1,000,000 generations by every crossover operator employed in the genetic algorithm, as described in Section 7.3. The instances are the same as in the previous analysis, and the crossovers are ordered with the same criterion.

In general, some considerations made for Table 4 are still valid for Table 5. In particular, hybrid crossovers based on the T path selection strategy are the slowest operators in both cases.

However, these computation times lie on a much smaller range: while in Table 4, the minimum value for each problem was 5-6 orders of magnitude smaller than the corresponding maximum value, the ratio between the maximum and the minimum values of Table 5 is only around two orders of magnitude.

The difference between the results in Tables 4 and 5 can be mainly explained by taking into account that during the execution, the average distance between the permutations tend to decrease, although with a rate strongly depending on the crossover, as we have seen at the end of Section 7.3. In fact, for all the algebraic crossovers a greater diversity means also a greater computation time, because the length of the minimal decomposition depends on the distance[8]. This greatly affects the algebraic crossovers which use the vertex selection strategies P and B, because they need to evaluate d solutions.

[8]It exactly correspond to the distance in the case of the ASW generating set

Table 5. Computational Times in the GA Experiment (in seconds)

Crossover	LOP	PFSP	QAP	TSP
PMX	7.0520	1.1700	4.1110	1.0510
OX1	8.3740	1.8410	4.1400	1.6260
AP	20.3840	2.1050	4.8890	1.2880
AXG-ASW-RR	17.8630	4.1510	16.5080	3.0010
POS	11.5050	4.7020	6.8780	5.2890
OX2	13.0360	4.9300	7.0640	5.2550
AXG-EXC-RR	18.6230	4.7900	16.0430	4.8790
AXG-ASW-RT	23.8080	4.3730	16.3130	3.6380
AXG-EXC-RB	9.3230	10.5670	37.9390	4.1030
AXG-INS-RB	10.6840	9.1310	37.9650	4.8430
AXG-ASW-RB	9.2150	13.3850	44.1530	3.3680
AXG-ASW-RP	9.1190	14.3300	44.9560	4.7190
AXG-INS-RP	12.0590	8.8940	39.9100	4.9050
AXG-INS-RT	18.0910	5.9410	17.6270	4.9090
CX	17.4590	5.5750	8.5470	5.8750
AXG-INS-RR	19.4140	5.9360	17.5530	4.9280
AX-Comb	16.3790	14.5750	20.6440	4.8930
AXL-Meet	58.6510	9.6550	23.9360	3.3330
AXL-Join	60.9260	8.9120	23.6660	3.8950
AXG-EXC-RP	12.9610	11.4440	38.1540	5.8310
AXG-EXC-RT	29.9490	5.3490	16.2080	7.5220
AXG-INS-TR	28.2510	16.7880	79.5620	5.5840
AXG-INS-TB	19.6620	19.9400	100.9430	5.5490
AXG-INS-TP	20.7750	19.3490	98.2840	5.5710
AXG-INS-TT	28.4800	16.4600	81.3820	5.6320
AXG-ASW-TR	24.9460	28.1740	100.8530	5.4250
ER	68.6220	16.0100	29.9970	19.8760
AXG-ASW-TB	16.9630	34.2740	122.6110	5.7680
AXG-ASW-TP	16.9650	38.0100	120.3980	7.3450
AXG-EXC-TR	31.2340	23.7160	80.8070	7.8600
AXG-EXC-TB	19.8490	28.8810	104.1220	8.1930
AXG-EXC-TP	23.4960	28.7850	104.4550	9.4060
AXG-EXC-TT	40.1330	28.5370	83.9950	12.6260
AXG-ASW-TT	31.0810	33.4880	121.1390	6.9190
AXH-RB	121.5390	151.8730	70.5480	66.7300
AXH-RR	131.8670	44.2730	76.5690	64.2880
AXH-RP	127.8180	147.6200	72.9290	79.0170
AXH-RT	142.5050	44.9490	75.4470	74.4410
AXH-TP	137.2240	383.0650	90.4920	96.9840
AXH-TB	123.6060	375.1560	108.4110	85.7040
AXH-TR	147.0350	275.1190	115.8140	79.9230
AXH-TT	141.4670	278.6580	131.3870	106.0500

8. Conclusion and Future Work

In this paper we have presented 34 crossover operators divided in three families: the group-based, the lattice-based and the hybrid algebraic crossover operators. We have shown their properties, focusing on the precedence relations which are transmitted from parents to their offspring.

To the best of our knowledge, this is the first study that consider the rich algebraic structure of the permutation space in the design of crossover operators for evolutionary algorithms.

These new crossovers have been experimentally compared with 7 existing crossover operators by conducting three series of experiments. In the first experiment (on random permutations) our crossover operators outperform their competitors, in the second test (on local optima) some of the group-families operators performed better than the other crossover operators. In the third experiment, we have shown that some of our crossover operators based on the INS generating set outperformed all the competitors.

In general, the best crossovers in our proposal always outperform the existing ones, therefore we can conclude that the experimental investigation carried out in this article clearly shows that the algebraic crossovers are a valid and effective alternative to the classical operators available in the literature.

The results obtained in this paper are promising and we are planning to apply these crossover operators to some specific combinatorial optimization problems and with a well suited algorithm in order to see if it is possible to reach state-of-the-art performances.

This approach to design group-based algebraic crossover operators can also be extended to other finitely generated groups, like those of bitstrings [17].

The lattice crossover operators can also be defined for other lattices, like the binary trees, which do not form a finitely generated group. It would be interesting to investigate their usage in genetic or other evolutionary algorithms.

Finally, another line of research is to investigate if the existing crossover in literature satisfies some of the algebraic properties of crossovers here introduced.

References

[1] Michalewicz Z, Hartley SJ. Genetic algorithms+ data structures= evolution programs. Mathematical Intelligencer, 1996. **18**(3):71.

[2] Milani A, Santucci V. Asynchronous Differential Evolution. In: Proc. of 2010 IEEE Congress on Evolutionary Computation (CEC 2010). 2010 pp. 1–7. doi:10.1109/CEC.2010.5586107.

[3] Umbarkar A, Sheth P. Crossover Operators in Genetic Algorithms: a Review. ICTACT journal on soft computing, 2015. **6**(1).

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

[5] Whitley D, Hains D, Howe A. Tunneling between optima: partition crossover for the traveling salesman problem. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation. ACM, 2009 pp. 915–922.

[6] Martí R, Reinelt G. The linear ordering problem: exact and heuristic methods in combinatorial optimization, volume 175. Springer Science & Business Media, 2011.

[7] Santucci V, Ceberio J. Using pairwise precedences for solving the linear ordering problem. Applied Soft Computing, 2020. **87**. doi:https://doi.org/10.1016/j.asoc.2019.105998.

[8] Santucci V, Baioletti M, Milani A. Algebraic Differential Evolution Algorithm for the Permutation Flowshop Scheduling Problem With Total Flowtime Criterion. IEEE Transactions on Evolutionary Computation, 2016. **20**(5):682–694. doi:10.1109/TEVC.2015.2507785.

[9] Baioletti M, Milani A, Santucci V. Algebraic Crossover Operators for Permutations. In: Proc. of 2018 IEEE Congress on Evolutionary Computation (CEC 2018). 2018 pp. 1–8. doi:10.1109/CEC.2018.8477867.

[10] Baioletti M, Milani A, Santucci V. Algebraic Particle Swarm Optimization for the permutations search space. In: Proc. of 2017 IEEE Congress on Evolutionary Computation (CEC 2017). 2017 pp. 1587–1594. doi:10.1109/CEC.2017.7969492.

[11] Baioletti M, Milani A, Santucci V. Automatic Algebraic Evolutionary Algorithms. In: Proc. of International Workshop on Artificial Life and Evolutionary Computation (WIVACE 2017). 2018 pp. 271–283. doi:10.1007/978-3-319-78658-2_20.

[12] Baioletti M, Milani A, Santucci V. Learning Bayesian Networks with Algebraic Differential Evolution. In: Proc. of 15th International Conference on Parallel Problem Solving from Nature (PPSN XV). 2018 pp. 436–448. doi:10.1007/978-3-319-99259-4_35.

[13] Burkard RE, Karisch SE, Rendl F. QAPLIB–a quadratic assignment problem library. Journal of Global optimization, 1997. **10**(4):391–403.

[14] Schiavinotto T, Stützle T. A review of metrics on permutations for search landscape analysis. Computers & Operations Research, 2007. **34**(10):3143–3153.

[15] Baioletti M, Milani A, Santucci V. An Extension of Algebraic Differential Evolution for the Linear Ordering Problem with Cumulative Costs. In: Proc. of 14th Int. Conf. on Parallel Problem Solving from Nature (PPSN XIV). 2016 pp. 123–133. doi:10.1007/978-3-319-45823-6_12.

[16] Baioletti M, Milani A, Santucci V. MOEA/DEP: An Algebraic Decomposition-Based Evolutionary Algorithm for the Multiobjective Permutation Flowshop Scheduling Problem. In: Evolutionary Computation in Combinatorial Optimization. 2018 pp. 132–145. doi:10.1007/978-3-319-77449-7_9.

[17] Baioletti M, Milani A, Santucci V, Tomassini M. Search Moves in the Local Optima Networks of Permutation Spaces: The QAP Case. In: Proc. of the 2019 Genetic and Evolutionary Computation Conference Companion (GECCO 2019). 2019 p. 15351542. doi:10.1145/3319619.3326849.

[18] Baioletti M, Milani A, Santucci V. Variable neighborhood algebraic Differential Evolution: An application to the Linear Ordering Problem with Cumulative Costs. Information Sciences, 2020. **507**:37–52. doi:10.1016/j.ins.2019.08.016.

[19] Herstein IN. Abstract Algebra, 3rd Edition. Wiley & Sons, 1996. ISBN 978-0-471-36879-3.

[20] Bespamyatnikh S, Segal M. Enumerating longest increasing subsequences and patience sorting. Information Processing Letters, 2000. **76**(1–2):7–11. doi:http://dx.doi.org/10.1016/S0020-0190(00)00124-1.

[21] Baioletti M, Milani A, Santucci V, Bartoccini U. An Experimental Comparison of Algebraic Differential Evolution Using Different Generating Sets. In: Proc. of the Genetic and Evolutionary Computation Conference Companion (GECCO 2019). 2019 p. 15271534. doi:10.1145/3319619.3326854.

[22] Grätzer G, Wehrung F. Lattice Theory: Special Topics and Applications - Vol. 2. Springer, 2016.

[23] Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review, 1999. **13**(2):129–170.

[24] Andreica A, Chira C. Best-order crossover for permutation-based evolutionary algorithms. Applied Intelligence, 2015. **42**(4):751–776. doi:10.1007/s10489-014-0623-0.

[25] Goldberg DE, Lingle R, et al. Alleles, loci, and the traveling salesman problem. In: Proceedings of an international conference on genetic algorithms and their applications, volume 154. Lawrence Erlbaum, Hillsdale, NJ, 1985 pp. 154–159.

[26] Davis L. Applying Adaptive Algorithms to Epistatic Domains. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1, IJCAI'85. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN 0-934613-02-8, 1985 pp. 162–164. URL `http://dl.acm.org/citation.cfm?id=1625135.1625164`.

[27] Syswerda G. Schedule Optimization Using Genetic Algorithms. In: Handbook of Genetic Algorithms. 1991 pp. 332–349.

[28] Oliver IM, Smith DJ, Holland JRC. A Study of Permutation Crossover Operators on the Traveling Salesman Problem. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application. L. Erlbaum Associates Inc., Hillsdale, NJ, USA. ISBN 0-8058-0158-8, 1987 pp. 224–230. URL `http://dl.acm.org/citation.cfm?id=42512.42542`.

[29] Larrañaga P, Kuijpers CMH, Poza M, Murga RH. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. Statistics and Computing, 1997. **7**(1):19–34. doi:10.1023/A:1018553211613. URL `https://doi.org/10.1023/A:1018553211613`.

[30] WHITLEY D. Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Reconbination. Handbook of Genetic Algorithms, 1991. pp. 350–372.

[31] Gavanelli M, Nonato M, Peano A, Alvisi S, Franchini M. Scheduling Countermeasures to Contamination Events by Genetic Algorithms. AI Commun., 2015. **28**(2):259–282. URL `http://dl.acm.org/citation.cfm?id=2733572.2733579`.

[32] Gopal R, Rosmaita B, Van Gucht D. Genetic algorithms for the traveling salesman problem. In: Proc. of 1st International Conference on Genetic Algorithms and their Applications. 1985 pp. 160–165.

[33] Schiavinotto T, Sttzle T. The Linear Ordering Problem: Instances, Search Space Analysis and Algorithms. J. Math. Model. Algorithms, 2004. **3**:367–402. doi:10.1023/B:JMMA.0000049426.06305.d8.

[34] Glover F, Laguna M, Martí R. Fundamentals of scatter search and path relinking. Control and cybernetics, 2000. **29**(3):653–684.

[35] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. of IEEE Intern. Conf. on Neural Networks, volume 4. 1995 pp. 1942–1948.

[36] Poli R, Kennedy J, Blackwell T. Particle swarm optimization: An overview. Swarm Intelligence, 2007. **1**(1):33–57.

[37] del Valle Y, Venayagamoorthy GK, Mohagheghi S, Hernandez JC, Harley RG. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. IEEE Transactions on Evolutionary Computation, 2008. **12**(2):171–195.

[38] Kennedy J, Eberhart RC. A discrete binary version of the particle swarm algorithm. In: Proc. of IEEE International Conference on Systems, Man, and Cybernetics, volume 5. 1997 pp. 4104–4108.

[39] Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research, 2007. **177**(3):1930–1947.

[40] Ai TJ, Kachitvichyanukul V. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research, 2009. **36**(5):1693–1702.

[41] Koulinas G, Kotsikas L, Anagnostopoulos K. A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem. Information Sciences, 2014. **277**:680–693.

[42] Gao H, Kwong S, Fan B, Wang R. A Hybrid Particle-Swarm Tabu Search Algorithm for Solving Job Shop Scheduling Problems. IEEE Transactions on Industrial Informatics, 2014. **10**(4):2044–2054.

[43] Cagnina L, Esquivel S, Gallard R. Particle swarm optimization for sequencing problems: a case study. In: Proc. of Congress on Evolutionary Computation, volume 1. 2004 pp. 536–541.

[44] Bean JC. Genetic Algorithms and Random Keys for Sequencing and Optimization. ORSA Journal on Computing, 1994. **6**(2):154–160.

[45] Bratton D, Kennedy J. Defining a Standard for Particle Swarm Optimization. In: Proc. of IEEE Swarm Intelligence Symposium. 2007 pp. 120–127.

[46] Hutter F, Hoos HH, Leyton-Brown K. Sequential Model-Based Optimization for General Algorithm Configuration. In: Proc. of LION-5 (Learning and Intelligent Optimization Conference). 2011 pp. 507–523.

[47] Birattari M. Tuning Metaheuristics: A Machine Learning Perspective. Springer, 2009.

[48] 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 Computation, 2011. **1**(1):3–18.

[49] Baioletti M, Milani A, Santucci V. A discrete differential evolution algorithm for multi-objective permutation flowshop scheduling. Intelligenza Artificiale, 2016. **10**(2):81–95.

[50] Ayodele M, McCall JAW, Regnier-Coudert O. RK-EDA: A Novel Random Key Based Estimation of Distribution Algorithm. In: Parallel Problem Solving from Nature - PPSN XIV - 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings. 2016 pp. 849–858.

[51] Santucci V, Baioletti M, Di Bari G, Milani A. A Binary Algebraic Differential Evolution for the MultiDimensional Two-Way Number Partitioning Problem. In: Evolutionary Computation in Combinatorial Optimization. 2019 pp. 17–32. doi:10.1007/978-3-030-16711-0_2.