Algebraic Crossover Operators for Permutations

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

***Abstract*—Crossover operators are very important tools 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 14 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.** [1]

***Index Terms*—Permutation crossovers, Algebraic crossovers, Lattice operators.**

I. Introduction

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

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], [2] have natural applications when the solutions are represented by, respectively, binary strings and real-valued vectors, while the edge assembly crossover [3] and the partition crossover [4] 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 [5] and the permutation flowshop scheduling problem [6].

The main contribution of this paper is the definition of three new families of algebraic crossover operators for the permutation space.

[1] **This paper has been accepted for presentation at next IEEE Congress on Evolutionary Computation (IEEE CEC 2018).**

The first family is mainly based on the algebraic framework we have introduced in [6], [7]. In the previous papers 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 in order to define a set of six new group-based algebraic crossover operators.

A new algebraic property of the permutation space, i.e., the fact that permutations form a lattice structure, is exploited to provide two new lattice-based algebraic crossover operators.

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

Some properties of these new 14 crossovers are discussed, while their performances have been experimentally compared with seven popular permutation crossover from literature.

Three different experimental scenarios have been considered. Indeed, 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 held on 12 selected instances from the four most popular permutation problems, i.e., the linear ordering problem [5], the permutation flowshop scheduling [6], the quadratic assignment problem [8], and the traveling salesman problem [3].

The rest of the papers is organized as follows. Section II briefly recalls the algebraic framework for the permutation space. The group-based, lattice-based and hybrid algebraic crossover families are introduced in, respectively, Sections III, IV and V. The seven permutation crossover from literature used in our experimentation are briefly described in Section VI. The experimental analysis is provided in Section VII, while conclusions are drawn in Section VIII, where future research directions are also proposed.

II. Algebraic Background

Many combinatorial optimization problems, like for instance the Traveling Salesman Problem (TSP) and Quadratic Assignment Problem (QAP), have the aim of optimizing 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 [6]. These concepts are used in the later sections to introduce the algebraic crossover operators.

A. 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 identiy 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 of $\mathcal{S}_n$ elements, called *generators*, such that any permutation of $\mathcal{S}_n$ can be expressed as a product of finitely many generators. Actually, $\mathcal{S}_n$ has multiple generating sets. Here, we consider the generating set of all the $n-1$ simple transpositions, i.e., the set $ST = \{\sigma_i \in \mathcal{S}_n : 1 \leq i < n\}$ where σ_i is defined as: $\sigma_i(i) = i + 1$, $\sigma_i(i + 1) = i$, and $\sigma_i(j) = j$ for $j \in [n] \setminus \{i, i + 1\}$.

Generally, any permutation has multiple decompositions in terms of ST. 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.

Finally, it is worth to note that $|x|$ 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 $i \neq j \in [n]$. We denote by $I(x)$ the set of all the inversions of x.

B. Cayley graphs

An important concept for any finitely generated group is its Cayley graph. For $\mathcal{S}_n$ generated by ST, the Cayley graph $\mathcal{CG}$ is the labeled digraph whose vertexes are the elements of $\mathcal{S}_n$ and there is an arc from x to y labeled by $\sigma_i \in ST$ if and only if $y = x \circ \sigma_i$. Hence, since it is easy to show that applying a simple transpositions corresponds to swap two adjacent items, $\mathcal{CG}$ actually represents the permutation search space equipped with the neighborhood structure induced by adjacent swap moves.

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

In $\mathcal{CG}$, 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 $(\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_L})$, then $x = \sigma_{j_1} \circ \sigma_{j_2} \circ \cdots \circ \sigma_{j_L}$. 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 $(\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_L})$, then $y = x \circ (\sigma_{j_1} \circ \sigma_{j_2} \circ \cdots \circ \sigma_{j_L})$, hence $\sigma_{j_1} \circ \sigma_{j_2} \circ \cdots \circ \sigma_{j_L} = x^{-1} \circ y$.

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-τ distance [9].

Finally, we write $x \sqsubseteq y$ if, for each minimal decomposition $s_x \in ST^*$ of x, there exists a minimal decomposition $s_y \in ST^*$ 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. It is known that $x \sqsubseteq y$ if and only if $I(x) \subseteq I(y)$ (see for instance [10]).

C. Vector operations

In a previous series of papers [6], [7], [11] 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 $\mathcal{S}_n$ generated by ST.

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 $\mathcal{CG}$, 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 $s_y = (\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_L})$ 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., $z = x \circ (\sigma_{j_1} \circ \sigma_{j_2} \circ \cdots \circ \sigma_{j_L})$, which simply reduces to

$$x \oplus y := x \circ y. \tag{1}$$

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 $(\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_L})$ in a shortest path from y to x. Since $\sigma_{j_1} \circ \sigma_{j_2} \circ \cdots \circ \sigma_{j_L} = y^{-1} \circ x$, we define $\ominus$ as

$$x \ominus y := y^{-1} \circ x. \tag{2}$$

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

Given a coefficient $a \in [0, 1]$, $a \odot x$ is defined for a vector x and produces a vector z 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, truncating it after $\lceil a \cdot |x| \rceil$ generators, and composing the truncated sequence. Since minimal decompositions are not unique, $\odot$ is a stochastic operators.

Since permutations composition and inversion can be computed in $O(n)$ time steps, also $\oplus$ and $\ominus$ have $O(n)$ complexity. The operation $\odot$ uses the $RandBS$ algorithm, introduced in

[6], [12], which produces a random minimal decomposition of a permutation in terms of simple transpositions. $RandBS$ 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

```
1: function RANDBS(x ∈ S_n)
2:     s ← ⟨⟩
3:     A ← {σ_i ∈ ST : i < i + 1 and x(i) > x(i + 1)}
4:     while A ≠ ∅ do
5:         σ ← select a generator from A uniformly at random
6:         x ← x ∘ σ
7:         s ← Concatenate(⟨σ⟩, s)
8:         A ← Update(A, σ)                    ▷ O(1) complexity
9:     end while
10:    return s
11: end function
```

Fig. 1. Randomized decomposition algorithms for permutations

iteratively choosing a random adjacent swap moves from the set of adjacent inversions 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 $(i+1,i+2)$. Since the computational cost of $RandBS$ is $O(n^2)$, so is the cost of computing $\odot$.

III. Group-based Algebraic Crossover Operators

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

Given two permutations $x, y \in \mathcal{S}_n$, the interval $[x, y]$ can be defined in various equivalent ways:

$$[x,y] = \{z \in \mathcal{S}_n \,:\, \exists p \in SP_{x,y} \text{ s.t. } z \text{ appears in } p\}, \quad (3)$$

$$[x,y] = \{z \in \mathcal{S}_n \,:\, z \ominus x \sqsubseteq y \ominus x\}, \quad (4)$$

$$[x,y] = \{z \in \mathcal{S}_n : d(x,z) + d(z,y) = d(x,y)\}. \quad (5)$$

$$[x,y] = \{z \in \mathcal{S}_n \,:\, D(x,z) \subseteq D(z,y)\}. \quad (6)$$

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 generating set ST.

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 $z \in [x, y]$.

This definition has an important interpretation in terms of precedences among the items in $[n]$. Indeed, given $x, y, z \in \mathcal{S}_n$, $z \in [x, y]$ if and only if

$$P(x) \cap P(y) \subseteq P(z) \subseteq P(x) \cup P(y), \quad (7)$$

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 (7) 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 (3).

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

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

Hence, we say that AXG is *precedence respectful* (property 1) and *transmits precedences* (property 2). Note also that, since $[x, x] = \{x\}$, then $AXG(x, x) = x$ for any possible implementation of AXG.

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 the enumeration of 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.

Two strategies are devised for the shortest-path construction phase: random (R), and greedy (G) construction. Three strategies are considered for the vertex selection phase: uniformly random (R), based on path truncation (T), and choosing the best vertex in the path (B). Hence, six AXG implementations are derived from any possible combination of the shortest-path and vertex selection strategies.

Given $x, y \in \mathcal{S}_n$, a random shortest path p connecting x to y (strategy R) is generated in two steps. First, obtain a random minimal decomposition $(\sigma_{j_1}, \sigma_{j_2}, \ldots, \sigma_{j_L})$ of $y \ominus x$ by using $RandBS$. Then, convert the minimal decomposition to the sequence of vertexes $p = (z_0, z_1, \ldots, z_L)$ such that $z_0 = x$ and $z_k = z_{k-1} \circ \sigma_{j_k}$ for all $1 \leq k \leq L$. It is easy to verify that $z_L = y$. An argumentation provided in [6] can be used to show that this procedure guarantees the largest degree of randomness without increasing the $O(n^2)$ asymptotic complexity.

The greedy construction procedure for a shortest path (strategy G) is very similar to the random one. The only modification is that, in the first step, $RandBS$ is replaced with the greedy decomposer $GreedyBS$, whose pseudo-code is provided in Figure 2.

$GreedyBS$ differs from $RandBS$ at line 5, where the generator $\sigma \in A$ is greedily chosen by means of the objective function f. $GreedyBS$ requires $O(n)$ fitness evaluations at each iteration. Since the maximum number of iterations is $O(n^2)$, the overall number of solutions to be evaluated is $O(n^3)$. Note anyway that, in most problems like, for instance, LOP and TSP, there are speed-up techniques that allow to compute in constant time the fitness of a solution x' obtained by applying an adjacent swap to x, being known $f(x)$.

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

```
1: function GREEDYBS(x ∈ S_n, f : S_n → ℝ)
2:     s ← ⟨⟩
3:     A ← {σ_i ∈ ST : i < i + 1 and x(i) > x(i + 1)}
4:     while A ≠ ∅ do
5:         σ ← select σ ∈ A with the best value of f(x ∘ σ)
6:         x ← x ∘ σ
7:         s ← Concatenate(⟨σ⟩, s)
8:         A ← Update(A, σ)              ▷ done in O(1) complexity
9:     end while
10:    return s
11: end function
```

Fig. 2. Greedy decomposition algorithms for permutations

```
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 ∈ ST : x'(i) > x'(i + 1) and y'(i) > y'(i + 1)}
6:     while A ≠ ∅ do
7:         σ ← select a generator fromA
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
```

Fig. 3. Meet operator for permutations

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 select the $\lceil \alpha \cdot L \rceil$-th solution from p. The best strategy B returns the fittest solution in p excluding again the two end-points. This last strategy requires the evaluation of solutions in the path (when the path is chosen randomly). However, the same speed-up techniques mentioned for *GreedyBS* can be also applied here.

Therefore, we have six group-based algebraic crossovers denoted by: *AXG-RR*, *AXG-RT*, *AXG-RB*, *AXG-GR*, *AXG-GT*, and *AXG-GB* (the first and second letters after the hyphen denote, respectively, the shortest path and vertex selection strategies). The crossovers *AXG-RR* and *AXG-GB* are polynomial approximations of, respectively, the most explorative and exploitative way to choose $z \in [x, y]$. *AXG-GB* is also related to the path-relinking operator [13]. $AXG\text{-}RT(x, y; \alpha)$ produces an offspring z such that $z = x \oplus \alpha \odot (y \ominus x)$. Both *AXG-RT* and *AXG-GT*, when applied with $\alpha = 0.5$, produce an offspring as far as possible from both parents (in terms of the Kendall-τ distance), thus they are likely to slow down the loss diversity with respect to the other proposals. Finally, note that *AXG-RB* and *AXG-GR* balance exploration and exploitation in two "orthogonal" ways.

IV. Lattice-based Algebraic Crossover Operators

The permutation space $\mathcal{S}_n$ endowed with the partial order relation $\sqsubseteq$ forms a lattice, i.e., it admits the two well defined binary operations of meet and join [10]. 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.

In this paper we also propose to employ the meet and join as crossover operators denoted by, respectively, *AXL-Meet* and *AXL-Join*.

The operator *AXL-Meet* can be computed by an algorithm similar to *RandBS* and *GreedyBS*. Its pseudo-code is provided in Figure 3. *AXL-Meet* initializes z to the identity. Then, at every iteration k, it moves z 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 Section II-A).

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-Join* by means of the following equivalence

$$x \vee y = \left(x^R \wedge y^R\right)^R, \tag{8}$$

and thus reusing the *AXL-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 $I(x \wedge y) \subseteq I(x) \cap I(x)$ and $I(x \vee y) \supseteq I(x) \cup I(y)$ (see for instance [10]), it is possible to show that

$$[x, y] \subseteq [x \wedge y, x \vee y], \tag{9}$$

where the equivalence holds if and only if $I(x \wedge y) = I(x) \cap I(y)$ and $I(x \vee y) = I(x) \cup I(y)$. As a special case, equivalence holds when x and y are comparable. However, it is important to observe that, conversely from the group-based crossovers, property (9) implies that meet and join can generate an offspring with new precedences with respect to x and y.

V. Hybrid Algebraic Crossover Operators

In this section we introduce the family of crossovers *AXH* which hybridizes the group-based family *AXG* (see Section III) with the lattice-based operators (see Section IV).

This hybrid family is motivated from property (9). Given $x, y \in \mathcal{S}_n$, we define $AXH(x, y)$ as a 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 III, we define six variants of hybrid algebraic crossovers as follows:

$$\begin{aligned}
AXH\text{-}RR(x,y) &= AXG\text{-}RR(x \wedge y, x \vee y),\\
AXH\text{-}RT(x,y;\alpha) &= AXG\text{-}RT(x \wedge y, x \vee y;\alpha),\\
AXH\text{-}RB(x,y) &= AXG\text{-}RB(x \wedge y, x \vee y),\\
AXH\text{-}GR(x,y) &= AXG\text{-}GR(x \wedge y, x \vee y),\\
AXH\text{-}GT(x,y;\alpha) &= AXG\text{-}GT(x \wedge y, x \vee y;\alpha),\\
AXH\text{-}GB(x,y) &= AXG\text{-}GB(x \wedge y, x \vee y).
\end{aligned}$$

VI. Permutation Crossovers in Literature

In this section we review some popular permutation crossover operators available in literature. This review is mainly based on [2], [14], [15].

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 [16] 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 $OX1$ [17] 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 [18]. 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 [19] 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 $z(j) = y(j)$, where j is the position of $x(1)$ in y (i.e. $y(j) = x(1)$). 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 [20] 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 [21] 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 [18] 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.

VII. Experiments

Aiming to compare the performances of the proposed crossover operators with those described in Section VI, experiments have been held 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. The name of the instances, together with the objective function formulations, are provided in Table I. All the selected instances come from widely adopted benchmark suites: xLOLIB[2] for LOP, Taillard's benchmark suite[3] for PFSP, QAPLIB[4] for QAP, and TSPLIB[5] for TSP. Moreover, PFSP has been investigated using the total flowtime as objective criterion [6], while, for TSP, we have adopted the objective function formulation that fixes the last city in the tour [22], thus allowing a one-to-one correspondence between TSP tours and permutations of $n-1$ cities.

All the algebraic crossovers using the truncated vertex selection strategy have been investigated by setting $\alpha = 0.5$.

The 14 algebraic crossovers (see Sections III, IV, and V) and the 7 competitors (see Section VI) have been compared by means of three different experiments: (i) without embedding them in any algorithm and considering randomly generated parent solutions, (ii) as in the previous point, but considering local optima solutions, (iii) embedding them in a genetic algorithm. These three experiments are described in, respectively, Sections VII-A, VII-B, and VII-C.

A. 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 for all the considered crossover operators. The 21 offsprings are then evaluated and ordered by their fitness, thus the rank obtained by every crossover is averaged over all the 5000 pairs of parents. These average ranks, aggregated on every problem for the sake of clarity, are provided in Table II. The crossovers are ordered according to their overall average rank.

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

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

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

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

TABLE I
BENCHMARK PROBLEMS AND INSTANCES

Problem	Objective Function	Definition of symbols	Instances
LOP	$\max_{\pi \in \mathcal{S}_n} \left\{ \sum_{i=1}^{n} \sum_{j=i+1}^{n} M_{\pi(i),\pi(j)} \right\}$	M is the $n \times n$ I/O matrix	N-be75eec_150 N-stabu1_150 N-t59b11xx
PFSP	$\min_{\pi \in \mathcal{S}_n} \left\{ \sum_{i=1}^{n} c_{\pi(i),m} \right\}$	$c_{\pi(i),j} = p_{\pi(i),j} + \max\left\{c_{\pi(i-1),j}, c_{\pi(i),j-1}\right\}$ $c_{i,0} = c_{0,j} = 0$ $p_{i,j}$ is the processing time of job i on machine j n and m are the number of jobs and machines	tai100_5_0 tai100_10_0 tai100_20_0
QAP	$\min_{\pi \in \mathcal{S}_n} \left\{ \sum_{i=1}^{n} \sum_{j=1}^{n} F_{i,j} D_{\pi(i),\pi(j)} \right\}$	F is the $n \times n$ flow matrix D is the $n \times n$ distance matrix	lipa90a sko100a tai100a
TSP	$\min_{\pi \in \mathcal{S}_{n-1}} \left\{ \sum_{i=1}^{n-2} d_{\pi(i),\pi(i+1)} + d_{\pi(n-1),n} + d_{n,\pi(1)} \right\}$	$d_{i,j}$ is the distance between cities i and j n is the number of cities	kroA100 bier127 pr152

TABLE II
AVERAGE RANKS IN THE RANDOM EXPERIMENT

Crossover	LOP	PFSP	QAP	TSP	Overall
AXH-GB	**1.12**	**1.55**	**1.33**	**1.12**	**1.28**
AXG-GB	3.04	2.74	1.70	5.40	3.22
AXH-RB	7.39	3.92	5.83	3.25	5.10
AXG-RB	8.10	5.10	5.76	7.78	6.69
AXH-GT	3.90	11.91	5.79	5.50	6.78
AXH-GR	5.52	10.14	6.62	5.39	6.92
AXG-GT	5.94	9.15	5.97	10.75	7.95
AXG-GR	7.75	8.77	6.10	10.55	8.29
AXL-$Meet$	12.54	11.21	16.53	8.50	12.20
AXH-RR	14.46	14.14	15.38	10.96	13.74
AXH-RT	14.86	13.70	14.51	11.88	13.74
AXL-$Join$	16.53	15.79	16.29	8.44	14.26
ER	14.09	13.42	14.35	15.58	14.36
$OX1$	14.35	13.54	14.31	15.74	14.49
CX	14.41	13.69	14.33	15.75	14.55
$OX2$	14.45	13.60	14.44	15.75	14.56
AP	14.52	13.70	14.34	15.72	14.57
PMX	14.42	13.74	14.36	15.74	14.57
POS	14.46	13.74	14.38	15.70	14.57
AXG-RR	14.53	13.74	14.35	15.73	14.59
AXG-RT	14.61	13.70	14.33	15.75	14.60

Table II clearly shows that almost all the algebraic crossovers outperform the competitors. The only exceptions are AXG-RR and AXG-RT that have anyway a comparable overall average rank with respect to best performing competitor ER. In particular, the two most exploitative algebraic operators AXH-GB and AXG-GB largely outperform all the other crossovers. Finally, it is worth to note that the hybrid algebraic variants (AXH-$*$) have systematically obtained a better result with respect to their group-based counterpart (AXG-$*$), thus validating the contribution of the two lattice operators.

B. Local Optima Experiment

Aiming to investigate the performances starting from good parent solutions, in this experiment the parents have been selected to be local optima.

Furthermore, the AXG crossovers have been slightly modified in this experiment. Indeed, in order to limit the possibility that the produced offspring falls in the basin of attraction of one of the parents, the vertex selection strategy of the AXG schemes has been constrained to the central half of the path (i.e., excluding the first and last quarters).

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 [9] 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 VII-A. The average ranks are provided in Table III, where the crossovers are ordered according to their overall rank.

The results of Table III are quite different from the ones observed in the random experiment. This is likely due to the improved quality of the parent solutions. In particular, the competitor crossovers look to perform better than in the random experiment, though AXG-GB, AXG-RB, and AXG-GR have anyway obtained much better results. Moreover, conversely from the previous analysis, the lattice operators perform poorly and the group-based variants (AXG-$*$) have systematically outperformed their hybrid counterparts (AXH-$*$). This is possibly explained by the greater exploration degree of the AXH crossovers that, in a local optima scenario, is probably not appropriate.

For the sake of completeness, local search has been also applied to the offspring solutions. The average ranks of these results are not provided because they do not show significant differences among the crossovers. However, an interesting data has been observed. The local search, when applied to the

TABLE III
AVERAGE RANKS IN THE LOCAL OPTIMA EXPERIMENT

Crossover	LOP	PFSP	QAP	TSP	Overall
AXG-GB	**1.28**	**1.31**	**1.22**	**1.29**	**1.28**
AXG-RB	1.78	1.82	1.91	1.78	1.82
AXG-GR	3.93	5.47	8.44	7.20	6.26
CX	8.31	6.12	4.43	10.52	7.35
PMX	9.38	6.84	6.44	6.80	7.37
AXG-GT	5.08	7.05	9.36	8.41	7.48
$OX2$	8.80	6.93	9.12	10.82	8.92
POS	8.80	7.00	9.04	10.87	8.93
$OX1$	10.22	10.84	11.97	4.08	9.28
AXH-GB	12.84	12.64	4.40	9.50	9.85
AP	6.03	7.11	16.33	16.58	11.51
AXG-RR	6.45	9.51	15.54	14.60	11.53
AXH-RB	15.49	12.48	9.93	11.65	12.39
AXG-RT	8.31	11.34	16.36	16.32	13.08
AXH-GT	14.19	16.23	9.70	14.90	13.76
AXH-GR	16.36	16.48	10.25	14.37	14.37
ER	20.60	19.67	16.27	3.99	15.13
AXH-RT	16.34	15.73	16.83	20.08	17.25
AXL-$Meet$	19.22	19.11	18.10	13.95	17.60
AXH-RR	17.61	17.36	17.37	19.38	17.93
AXL-$Join$	19.97	19.94	18.01	13.87	17.95

TABLE IV
AVERAGE RANKS IN THE GA EXPERIMENT

Crossover	LOP	PFSP	QAP	TSP	Overall
PMX	9.33	2.33	**1.33**	6.67	**4.92**
POS	6.67	**2.00**	2.33	9.33	5.08
AX-$Comb$	3.33	7.67	8.00	**2.33**	5.33
$OX2$	5.67	6.00	4.00	8.00	5.92
CX	**2.33**	3.00	7.67	11.67	6.17
AXG-GB	4.00	5.67	7.33	8.67	6.42
$OX1$	8.67	5.00	8.00	4.00	6.42
AXG-RB	5.33	7.33	11.33	10.00	8.50
AXG-GR	3.33	9.00	10.00	14.00	9.08
AXG-RR	8.33	8.33	14.00	13.33	11.00
AXH-RB	14.33	13.33	7.33	10.00	11.25
AXG-GT	9.00	12.67	7.67	18.00	11.84
AP	13.67	13.33	16.00	5.00	12.00
AXH-GB	15.33	16.67	11.00	11.00	13.50
AXG-RT	14.67	13.33	12.33	18.00	14.58
AXH-RR	16.67	15.00	15.00	11.67	14.59
AXH-GR	14.33	18.33	14.67	12.33	14.92
AXH-GT	16.33	15.67	16.00	16.00	16.00
AXH-RT	18.67	15.33	16.00	15.67	16.42
ER	20.00	20.00	21.33	11.67	18.25
AXL-$Join$	21.33	21.00	21.00	17.67	20.25
AXL-$Meet$	21.67	22.00	20.67	18.00	20.59

solutions produced by AXG-GB and AXG-RB, performs in average less than 3 iterations, thus meaning that the produced offspring are very close to a local optimum. In particular, in the LOP instances, AXG-GB has produced new local optima (with respect to the parents) in around the 5% of the trials.

C. Genetic Algorithm Experiment

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

A standard steady-state GA scheme has been considered by taking inspiration from the one used in [14]. 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 a given probability. 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 [9] with probability 0.05. The population size has been set to 50, and each algorithm execution terminates after $1\,000\,000$ iterations.

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

At a first look, some of the competitor crossovers seem to outperform our proposals. However, we have observed that on every problem instance the fitness values produced by the best 20 executions among all our crossover operators outperform all the competitors. Hence, we have designed a meta-crossover operator AX-$Comb$, that at each iteration, selects one of our 14 operators according to a simple adaptive strategy. For each $AX*$ crossover, a *karma* value is maintained. At each iteration, AX-$Comb$ 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.

Table IV shows that AX-$Comb$ reaches competitive performances with respect to its competitors. In particular, its performances are very good in the TSP and LOP instances.

We have further investigated the reasons of this behavior by observing the the Kendall-τ distances between the parent solutions at every iteration. This experiment shows that AXG-$*$ crossovers loss diversity very quickly with respect to the classical operators. The same observation, though with a smaller degree, is true also for the hybrid algebraic crossovers, while the lattice-based meet and join looks do not show a significant diversity loss.

VIII. CONCLUSION AND FUTURE WORK

In this paper we have presented 14 new 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.

These new crossovers have been experimentally compared with seven existing crossover operators by conducting three series of experiments. When applied to randomly generated permutations, our crossover clearly operators outperform the competitors. In the experiment with local optima parent permutations, the three most performing operators are the exploitative variants of the group-based algebraic crossovers. When the crossovers are embedded in a standard genetic algorithms, though no single variant of our proposal obtains top average performances, the experiments show that the best executions come anyway from different AX operators. This means that the GA schemes using the AX crossovers are not robust enough. This fact led us to define a combined operator which adaptively chooses a (possibly) different AX crossover at every iteration. The performances of the combined crossover have been shown to be competitive with the competitors.

The results obtained in this paper are promising and we are planning to apply the algebraic crossovers to some specific combinatorial problems and with a well suited algorithm in order to reach competitive performances.

A further line of research is to extend the design of the group-based algebraic crossovers to other generating sets for the permutation group and to other finitely generated groups like, for instance, the bit-strings.

Finally, the lattice-based crossovers are a first step towards a deeper exploitation of the lattice structure characterizing some interesting combinatorial search spaces. For instance, the space of binary trees, though not being a group, forms a lattice (see [10]).

REFERENCES

[1] Z. Michalewicz and S. J. Hartley, "Genetic algorithms+ data structures= evolution programs," *Mathematical Intelligencer*, vol. 18, no. 3, p. 71, 1996.

[2] A. Umbarkar and P. Sheth, "Crossover operators in genetic algorithms: a review," *ICTACT journal on soft computing*, vol. 6, no. 1, 2015.

[3] Y. Nagata and S. Kobayashi, "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 346–363, 2013.

[4] D. Whitley, D. Hains, and A. Howe, "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.

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

[6] V. Santucci, M. Baioletti, and A. Milani, "Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 5, pp. 682–694, 2016.

[7] M. Baioletti, A. Milani, and V. Santucci, "Algebraic particle swarm optimization for the permutations search space," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 1587–1594.

[8] R. E. Burkard, S. E. Karisch, and F. Rendl, "Qaplib–a quadratic assignment problem library," *Journal of Global optimization*, vol. 10, no. 4, pp. 391–403, 1997.

[9] T. Schiavinotto and T. Stützle, "A review of metrics on permutations for search landscape analysis," *Computers & Operations Research*, vol. 34, no. 10, pp. 3143–3153, 2007.

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

[11] M. Baioletti, A. Milani, and V. Santucci, "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*, 2016, pp. 123–133.

[12] V. Santucci, M. Baioletti, and A. Milani, "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*. Springer International Publishing, 2014, pp. 161–170.

[13] F. Glover, M. Laguna, and R. Martí, "Fundamentals of scatter search and path relinking," *Control and cybernetics*, vol. 29, no. 3, pp. 653–684, 2000.

[14] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[15] A. Andreica and C. Chira, "Best-order crossover for permutation-based evolutionary algorithms," *Applied Intelligence*, vol. 42, no. 4, pp. 751–776, Jun 2015.

[16] D. E. Goldberg, R. Lingle *et al.*, "Alleles, loci, and the traveling salesman problem," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 154. Lawrence Erlbaum, Hillsdale, NJ, 1985, pp. 154–159.

[17] L. Davis, "Applying adaptive algorithms to epistatic domains," in *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, ser. IJCAI'85. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1985, pp. 162–164. [Online]. Available: http://dl.acm.org/citation.cfm?id=1625135.1625164

[18] G. Syswerda, "Schedule optimization using genetic algorithms," in *Handbook of Genetic Algorithms*, 1991, pp. 332–349.

[19] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "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*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1987, pp. 224–230. [Online]. Available: http://dl.acm.org/citation.cfm?id=42512.42542

[20] P. Larrañaga, C. M. H. Kuijpers, M. Poza, and R. H. Murga, "Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms," *Statistics and Computing*, vol. 7, no. 1, pp. 19–34, Mar 1997. [Online]. Available: https://doi.org/10.1023/A:1018553211613

[21] D. Whitley, "Traveling salesman and sequence scheduling: Quality solutions using genetic edge reconbination," *Handbook of Genetic Algorithms*, pp. 350–372, 1991.

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