

# Geometric crossover for the permutation representation

Alberto Moraglio<sup>a,\*</sup> and Riccardo Poli<sup>b</sup>

<sup>a</sup>*School of Computing and Centre for Reasoning, University of Kent, Canterbury, UK*

<sup>b</sup>*School of Computer Science and Electronic Engineering, University of Essex, Colchester, UK*

**Abstract.** Geometric crossovers are a class of representation-independent search operators for evolutionary algorithms that are well-defined once a notion of distance over a solution space is defined. In this paper we explore the specialisation of geometric crossovers to the permutation representation analysing the consequences of the availability of more than one notion of distance. Also, we study the relations among distances and build a rational picture in which pre-existing recombination operators for permutations fit naturally. Lastly, we illustrate the application of geometric crossover to the Travelling Salesman Problem (TSP).

Keywords: Evolutionary algorithms, crossover, permutations, theory

## 1. Introduction

Darwinian evolution – one of the most intriguing and powerful mechanism of nature – has been a constant source of inspiration for researchers interested in search and optimisation for over three decades, leading to the formation of a large research community and a huge body of literature which go under the name of *Evolutionary Computation* (EC).

Evolutionary algorithms (EAs) [2, 13, 18] rely on surprisingly few ingredients of Darwinian evolution, namely:

*Inheritance:* individuals have a genetic representation (in nature, the chromosomes and the DNA) such that it is possible for the offspring of an individual to inherit some of the features of its parent.

*Variation:* the offspring are not exact copies of the parents. Instead, reproduction involves mechanisms

that create innovation, as new generations are born. Typically, variation is produced both through *mutations* of the genome and through the effect of sexually recombining the genetic material coming from the parents to obtain offspring chromosomes (*crossing over*).

*Selection:* individuals best adapted to the environment have longer life and higher chances of mating and spreading their genetic makeup.

In EAs, the notion of individual corresponds to a tentative solution to a problem of interest. The fitness of natural individuals corresponds to the objective function used to evaluate the quality of the tentative solutions in the computer. The genetic variation processes of mutation and recombination are seen as mechanisms (search operators) to generate new tentative solutions to the problem. Finally, natural selection is interpreted as a mechanism to promote the diffusion and mixing of the genetic material of individuals representing good quality solutions, and, therefore, having the potential to create even fitter individuals (better solutions).

---

\*Corresponding author. E-mail: moraglio@dei.uc.pt, A.Moraglio@kent.ac.uk (A. Moraglio); rpoli@essex.ac.uk (R. Poli).

Typically evolutionary algorithms have the following form [3]:

1. Initialise population and evaluate the fitness of each population member
2. Repeat until a stopping condition is satisfied
  - (a) Select sub-population for reproduction on the basis of fitness (*Selection*)
  - (b) Copy some of the selected individuals (*Reproduction*), perform *Crossover* or *Mutation* on the remaining selected individuals
  - (c) Evaluate the fitness of the new population.

Although a lot more goes on in natural evolution, these ingredients are in fact sufficient to obtain artificial systems with the ability to find high quality individuals.

For all the phases in a EA there exist many different realisations. For example, there are many alternative genetic representations for solutions, the most important being binary strings, real-valued vectors, permutations, and trees. Also particularly striking is the case of the genetic operators (crossover and mutation), where there are completely different classes of operators for different representations. Also, even focusing on one representation, there are almost as many alternative realisations of crossover and mutation in the literature as there are researchers working in EC.

This is certainly a healthy situation, but it makes it difficult to understand the difference and similarities between different evolutionary methods. This has led us to start a research programme attempting to unify alternative operators and representations under a single more general theory [15]. In particular, in [16] we defined the *geometric crossovers* (also known as topological crossovers), which are a class of *representation-independent* operators that are well-defined once a notion of distance over the solution set is defined. All these operators have in common the fact that the offspring they produce are *between* their parents.<sup>1</sup>

This simple definition has surprising implications, including a powerful way to do crossover design for *any* representation and the potential for the development of a general theory of evolutionary algorithms encompassing *all* representations. Alternative ways of

linking a notion of distance to recombination operators are possible and have been actively pursued (see e.g., [12]).

In this paper we will focus in particular on the *permutation representation* (see [10] for an introduction). This is one of the most-frequently used representations in evolutionary algorithms. Many combinatorial optimisation problems, including TSP and scheduling problems, are naturally cast using permutations.

When applied to permutations, traditional crossover operators can produce invalid offspring. So, researchers have proposed a variety of operators specifically designed for permutations. They range from general-purpose operators working reasonably well on a wide spectrum of problems, such as the Partially Matched Crossover [11], to specialised operators that work best on a specific class of problems [8], such as Edge Recombination Crossover for TSP [22].

In previous work [16] we have shown how geometric crossover generalises the notion of crossover for binary strings. Differently from the binary string case, for which a single natural distance (the Hamming distance) is defined, permutations allow for various notions of distance that are all equally natural. In this paper, we explore how our geometric framework applies to the permutation representation and in particular analyse the consequences of having more than one notion of distance available. We study the relations among distances and we build a rational picture in which pre-existing recombination operators fit naturally. As an important example, we also analyse in detail the application of geometric crossover to TSP.

The paper is organised as follows. In section 2, we introduce the geometric framework. Section 3 introduces various notions of distance for permutations. Section 4 focuses on distances based on permutation interpretation and discusses the difficulty with geometric crossovers based on such distances. Section 5 introduces various edit distances. Section 6 shows that geometric crossover is naturally suited to edit distances. Section 7 draws a parallel between “interpretation” distances and edit distances and suggests that geometric crossovers defined over edit distances can be thought as geometric crossovers defined over the corresponding interpretation distances. In section 8, we suggest that many pre-existing recombination operators for permutations are in fact geometric crossovers under different edit distances. Section 9 gives an analysis of geometric crossover for the TSP problem, discussing problems in its applicability and solutions. In section 10 we present our conclusions.

<sup>1</sup> Note, however, that the meaning of the term “between” depends on the representation and the distance chosen.

## 2. Geometric framework

A *configuration space*  $C$  is a pair  $(G, Nhd)$  where  $G$  is a set of configurations (genotypes) and  $Nhd : G \rightarrow 2^G$  is a neighbourhood function which maps every configuration in  $C$  to the set of all its neighbour configurations in  $C$  which can be obtained by applying any unitary *move* from a pre-specified set. The neighbourhood function is generally symmetric ( $y \in Nhd(x) \Leftrightarrow x \in Nhd(y)$ , which is to say moves are reversible) and connected (any configuration can be transformed into any other in a finite number of moves). The same configuration set may be associated with more than a configuration space if multiple neighbourhood functions are available. The neighbourhood function induces an undirected *neighbourhood graph* (or *neighbourhood structure*)  $W = (V, E)$ , where  $V$  is the set of vertices representing configurations and  $E$  is the set of edges representing the relationship of neighbourhood between configurations. The neighbourhood structure is symmetric and connected. Thus, this space is also a *metric space* provided with a *distance function*  $d$  induced by the neighbourhood function which assigns as distance between any two configurations the length of a shortest path linking them in the neighbourhood structure. Both  $Nhd$  and  $d$  identify univocally the structure of the space, so we can equivalently write  $C = (G, Nhd)$  or  $C = (G, d)$ . A *fitness landscape*  $F$  is a pair  $(C, f)$  where  $C = (G, d)$  is a configuration space and  $f : G \rightarrow R$  is a *fitness function* mapping configurations to their *fitness values*.

In [16] we have defined two *classes* of *representation-independent operators*, which are defined on the representation only indirectly via the notion of *distance* associated to the *landscape*: geometric mutation and geometric crossover. We give the main definitions and properties for geometric crossover below since these are the starting point for the work on permutations reported in this paper. Note that alternative ways of linking a notion of distance to mutation and recombination operators are possible and have been actively pursued (see e.g., [12]).

In a metric space  $(S, d)$  a line segment is the set of the form  $[x; y] = \{z \in S | d(x, z) + d(z, y) = d(x, y)\}$  where  $x, y \in S$  are called end-points of the segment. The line segment can be thought of as a function of the underlying metric  $d$ .

A  $g$ -ary genetic operator  $OP$  takes  $g$  parents  $p_1, p_2, \dots, p_g \in C$  and produces one offspring  $c \in C$  according to a given conditional probability distribution:

$$Pr\{OP(p_1, p_2, \dots, p_g) = c\} = f_{OP}(c | p_1, p_2, \dots, p_g)$$

**Definition 1** (*Image set*) The image sets

$Im[OP(p_1, p_2, \dots, p_g)]$  of a genetic operator  $OP$  are the sets of all possible offspring that may be produced by  $OP$  from the parents  $p_1, p_2, \dots, p_g$ .

**Definition 2** (*Geometric crossover*) A binary operator is a geometric crossover under the metric  $d$  if all offspring are in the segment between their parents for the distance  $d$ .

**Definition 3** (*Uniform geometric crossover*) Uniform geometric crossover  $UX$  is a geometric crossover where all  $z$  laying between parents  $x$  and  $y$  have the same probability of being the offspring:

$$f_{UX}(z|x, y) = \frac{\delta(z \in [x; y])}{|[x; y]|}$$

$$Im[UX(x, y)] = \{z \in S | f_{UX}(z|x, y) > 0\} = [x; y].$$

**Theorem 1** The structure over the configuration space  $C$  is equivalently identified by the set  $G$  of the configurations together with any one of the following elements:

1. the neighbourhood function  $Nhd$ , 2. the neighbourhood graph  $W = (V, E)$ , 3. the distance function  $d$ , 4. the uniform geometric crossover  $UX$ , 5. the set of all segments  $H$ .

**Corollary 1** Given a structure of the configuration search space in terms of neighbourhood function or distance function,  $UX$  is unique.

**Corollary 2** Given a representation, there are as many  $UX$  operators as notions of distance for the representation.

The following two properties apply to binary strings:

**Theorem 2** All mask-based crossover operators for binary strings are geometric crossovers under Hamming distance.

**Theorem 3** The uniform geometric crossover for the configuration space of binary strings endowed with Hamming distance is the traditional uniform crossover.

## 3. Distances between permutations

Differently from binary strings where a single, natural definition of distance, the Hamming distance, is normally used, for permutations many notions of distance are equally natural. This situation is further complicated by the fact that such distances relate to

each others in various ways with subtle dependencies. Further complication arises from the fact that representations related to permutations such as circular permutations and permutations with repetitions are normally treated as if they were permutations, which is incorrect as they allow for different notions of distance. For a survey on metrics on permutations see [7].

Distances for permutations may have different origins:

1. Notions of distance arising from the *interpretation* of permutations: these measure the distance between the objects represented by two permutations (phenotypes).
2. Notions of distance directly connected with the *syntax*: these distances measure how two permutations differ in their syntax (genotypes).
3. Notions of distance connected with the idea of mutation or *edit* distance: these distances measure the minimum number of moves necessary to transform a permutation into another by the application of a unitary move operator defined on the syntax of permutations.

These three types of distances are interdependent. For example, an edit distance is also a syntactic distance but a syntactic distance is not necessarily an edit distance as one can define a measure of syntactic similarity that is not defined on edit moves. In the following three sections, we study the connections between these three notions of distance and their suitability as bases for geometric crossover.

In principle, given *any* notion of distance over the solution set, the corresponding geometric crossover and geometric mutation operators are well-defined. What is a good distance to use as a basis for these operators then? A good distance is one that (i) gives rise to a fitness landscape which is easy to search for the specific search operator employed and that (ii) allows such operator to be implemented efficiently. The first point is a design issue. A rule of thumb to pick a distance as a basis for geometric crossover that is likely to produce good performance is to choose a distance that is meaningful for the interpretation of the solution representation (phenotype) most suited to the problem at hand. The second point is an implementation issue. What is a class of distances that would allow us to build an operational procedure that implements the geometric crossover associated with them? Here we elaborate on the latter.

Geometric crossover and mutation are well-defined for any notion of distance, whether based on syntax or

not. In other words, operators are well-defined independently from any aspect of the underlying representation. In practice, however, the genetic operators have to be implemented. In this paper we show that if they are not based on a notion of edit distance that links them tightly to the solution representation, they become difficult or even *impossible to implement efficiently*. To understand the reasons for this we need to elicit a *duality* relating geometry in the search space and underlying representation. The notion of edit distance arising from the syntax of configurations has a natural dual interpretation:

1. seen in the configuration space, it is a measure of *dissimilarity* between two syntactic objects
2. seen in the neighbourhood graph, it is a measure of *spatial remoteness* between points in the search space.

Importantly, an edit move is a *natural unit* for both dissimilarity of configurations and discrete geometry of the search space. As a consequence, when a search operator is defined by means of edit moves using geometric definitions involving an edit distance, its definition has also a natural corresponding syntactical interpretation. This is because the same operator can be equivalently defined operationally by means of suitable combination and aggregation of edit moves originating in the syntax of the parents to produce the syntax of the offspring.

For each representation and edit move definitions, this duality manifests itself in a different way. In the case of permutations, as we will see, picking offspring on shortest paths between parents is equivalent to picking elements on minimal sorting trajectories from one parent permutation to the other. This connection between the geometric notion of “belonging to a segment” and its syntactic dual of “being on a minimal sorting trajectory” is ultimately what allows geometric crossover to be *actually implemented* possibly in an efficient way. So, even if a geometric crossover is representation-independent, when dealing with its implementation the specific representation and the specific distance used make indeed the difference between a practical operator, that can be implemented, and a merely theoretical one, which may not admit any operational definition.

#### 4. Permutation interpretations and related distances

To fully understand distances for permutations, we cannot separate them from permutation interpretations. Permutations can be used to represent solutions to different types of problems for which different relations

among the elements in the permutation are relevant. There are three major interpretations of a permutation [1]. For example, in TSP, permutations represent tours and the relevant information is the *adjacency relation* among the elements of a permutation. In resource scheduling problems, permutations represent priority lists and the relevant information is the *relative order* of the elements of a permutation. In other problems, the important characteristic is the *absolute position* of the elements in the permutation.

Let us consider the permutation (345261) that might be produced by randomly shuffling the elements of the identity permutation (123456). If adjacency is important then the fact that elements 4 and 5 are adjacent is relevant as is the fact that elements 3 and 2 are not. If the important aspect is relative order then what is relevant is that 4 precedes 5 and that 3 precedes 2. If absolute order is important then the relevant point is that 3 is at position 1, 4 at position 2, etc.

For each permutation interpretation, it is possible to write a binary matrix that represents the relation among pairs of elements in the permutation associated with that specific interpretation. So, we can have a relative order matrix (*ROM*), an absolute position matrix (*APM*) and an adjacency matrix (*AM*). For example, for the permutation (345261) we have the following matrices:

$$ROM = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$APM = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$AM = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The matrix *ROM* describes the binary relation of strict relative order precedence among all elements in the permutation. In the matrix, there is an entry for each possible pair of elements of the permutation. The entry is 0 if the element associated with its row does not precede in the permutation the element associated with its column. Otherwise the entry is 1.

The matrix *APM* describes the binary relation of absolute position of the elements in the permutation. In the matrix, rows are associated with positions of the permutation, columns are associated with elements of the permutation. The entry is 0 if the associated element with its column is not at the position associated with its row. If it is, the entry is 1.

The matrix *AM* describes the binary relation of adjacency of the elements in the permutation. In the matrix, there is an entry for each possible pair of elements of the permutation. The entry is 0 if the elements associated with its row and column are not adjacent in the permutation. Otherwise the entry is 1.

It is possible to define three distance functions for permutations based on relative order matrices, absolute position matrices and adjacency matrices. The distance between two permutations is then the Hamming distance between their corresponding matrices in the three interpretations. We refer to these distances as *relative order distance* (ROD), *absolute position distance* (APD) and *adjacency distance* (AD).

For example, the ROM matrices associated with the permutations (345261) and (123456) are, respectively, the ROM matrix provided in the example above and a triangular matrix filled with 1 in all entries above the main diagonal, and 0 elsewhere. The ROD distance between these permutations is the Hamming distance between these two matrices, which is 16. That gives a measure of their diversity in terms of relative order of their elements. In [15], we have shown that ROD and APD are metrics for permutations, while AD is a pseudo-metric for permutations and a metric for reversible permutations.

Once we have these distances, we have a well-defined notion of segment for each of them. For example, the segment between two permutations  $p_1$  and  $p_2$  under ROD includes all the permutations whose ROM are between the ROMs of  $p_1$  and  $p_2$ . This means that this segment includes all those permutations whose relative order relation among their elements is compatible with the common relative order among the elements of  $p_1$  and  $p_2$ .

Continuing the example above, the permutation  $p_3 = (341256)$  is in the ROD-segment between  $p_1 =$

(345261) and  $p_2 = (123456)$ . This can be verified by computing the ROM for  $p_3$ , which is:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Since  $ROD(p_1, p_2) = 16$ ,  $ROD(p_1, p_3) = 8$  and  $ROD(p_3, p_2) = 8$  then  $p_3 \in [p_1, p_2]_{ROD}$ .

Geometric operators can be defined using these notions of distance. So we can *rigorously* define relative order geometric crossover (ROX) and mutation (ROM), absolute position geometric crossover (APX) and mutation (APM), and adjacency geometric crossover (AX) and mutation (MX). ROX transmits perfectly to the offspring permutations the common relative order relation among elements of parent permutations. APX transmits perfectly to the offspring permutations the common absolute positions of the elements of parent permutations. AX transmits perfectly to the offspring (reversible) permutations the common adjacency relation of the elements of parent (reversible) permutations.

Let us consider the ROX crossover. This crossover could be implemented by recombining the ROMs of the parent permutations using a simple extension to matrices of the traditional crossover for binary strings and converting the offspring ROM into the corresponding offspring permutation. The price to pay to recombine permutations in this way is that offspring matrices are not guaranteed to be valid ROMs corresponding to feasible permutations. Hence, one may need to deal with this form of infeasibility, for example, by using some form of repair mechanism. Note that in this case the repaired offspring is not guaranteed anymore to cover the common relative order of the parent permutations perfectly. Analogous considerations hold for the APX and AX crossovers.

The infeasibility problem arises from the fact that the distances considered in this section as basis for geometric crossover are not directly based on permutations, but on an auxiliary matrix representation. In the next section, we consider distances defined directly on the “syntax” of permutations, edit distances, for which the infeasibility problem does not exist.

## 5. Edit distances and mutations

### 5.1. Mutations for permutations

Various mutation operators have been defined for permutations. The most common are [1]:

*Inversion or 2-change (block-reversal)*: This operator selects two points along the permutation then reverses the segment between the points. It is particularly well-suited for the TSP and for all the problems that naturally admit a permutation representation in which adjacency among elements is important.

*Insert and block-transposition*: This operator selects one element and inserts it at some other position in the permutation. There is a variant: one selects two elements and then moves the second element before the first. This move is irreversible because it is not possible to return to the previous permutation by a single further application of the move. These operators are used in scheduling problems in which relative order of elements is important.

*Swap and adjacent swap (two-element swap)*: The swap operator selects and swaps two elements. The adjacent swap swaps two contiguous elements.

*Scramble*: This operator selects a sublist of elements of the permutation and randomly reorders them while leaving the other elements in the permutation in the same absolute position.

Since the notion of mutation is naturally connected with edit moves, it is also connected with edit distances for permutations. However, mutations and edit moves do not coincide: an edit move is a deterministic and unitary syntactic transformation, whereas a mutation is a non-deterministic operator with a given probability distribution. So, it does not always correspond to a single edit move (a mutation can generate offspring more than one edit move away from their parent).

A valid edit move must be symmetric and connected. All the mutation operators listed above, except for the irreversible variant of the insert mutation, are associated with edit moves that are symmetric and fully connected. For example, the inversion operator is symmetric (re-reversing the same sub-list produces the original permutation) and connected (by repeated reversions it is possible to reach any permutation from any other permutation). Also the adjacent swap operator is symmetric and connected (bubble sort based on adja-

cent swap is able to sort any permutation of elements). The same holds for the swap operator.

Therefore, we can talk of *reversal distance*, *transposition distance*, *swap distance*, *adjacent swap distance*, *scramble distance* and so on. Notice that there are a number of variations for each of these distances which result from imposing constraints on the edit move.

## 5.2. Relations between edit distances and interpretation distances

Depending on the interpretation of the permutation, the same mutation can be seen as a small change or a major change. For example, the inversion operator does a minimal change when one thinks of a permutation in terms of adjacency, but a major change when the same permutation is seen as a priority list (relative order).

A single mutation should represent a minimal change at a phenotypic level to be likely to give rise to a smooth fitness landscape, which, in turn, may lead to good performance [19]. According to this principle, there are three mutation operators that make a minimal change in a permutation, one for each interpretation. When the permutation is thought of as an adjacency relation then the minimal mutation operator is the inversion operator: while reversing the order of a sub-list, only two adjacency links (edges) are changed. When the permutation represents a relative order the minimal mutation operator is the adjacent swap operator that affects only the relative order of a pair of elements. When the absolute position of elements in the permutation is relevant, the minimal mutation operator is the swap operator that changes the absolute positions of only two elements.

## 6. Sorting crossovers

### 6.1. Edit distances and sorting crossovers

For each notion of edit distance there is a corresponding notion of geometric crossover. So, we can define many possible crossovers for permutations, each induced from a corresponding mutation. Since these are crossovers based on similar, but not identical, neighbourhood structures, they will tend to have similar behaviours. Not all geometric crossovers based on edit distances have efficient implementations though. Indeed, constraints on edit moves transform the

complexity of computing the distance, hence of implementing the corresponding crossover, from polynomial to NP-hard [20].

Because of the distance duality, a point on a segment between two permutations, under a given edit distance, is on a minimal sorting trajectory connecting the two permutations. This allows to actually implement such geometric crossovers based on edit distances by *sorting algorithms*. Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Others give rise to crossovers that are possible to implement efficiently (in polynomial time) only in an approximated way. Quite interestingly, bubble sort and insertion sort fit the definition of geometric crossover for, respectively, the adjacent swap distance and the swap distance. So, *ordinary sorting algorithms can actually be used as crossovers!*

Not all classic sorting algorithms can be used as a base for geometric crossover though. We can adopt only those algorithms that use the same move throughout the sorting, and that are guaranteed to sort always using the minimum number of move applications. Technically, these algorithms when applied to permutations solve the “minimal permutation sorting by  $x$  problem” [21] where  $x$  stands for the move used. Bubble sort, insertion sort and selection sort belong to this class of sorting algorithms, for which the sorting move, respectively adjacent swap, insertion and swap, is pre-specified and fixed over the whole execution of the algorithm. Some more effective sorting algorithms, such as quick sort, use different moves while progressing with the sorting, so they cannot be used as a basis for geometric crossovers.

In the following we highlight some important differences between sorting algorithms and crossover operators for permutations based on edit moves:

1. Sorting algorithms sort permutations into the fully ordered permutation (123...). Crossover operators sort one parent permutation toward the order of the other parent permutation, which can be any permutation. However, sorting algorithms can be easily adapted to sort any permutation into any other (by renaming the elements of both permutations).
2. Traditional sorting algorithms are designed to sort vectors of any type of objects coming with a well-defined notion of order between objects: they can sort permutations, as well as real vectors, list of words, etc. Crossover operators sort permutations, which are special in that the ordering rank of each

element is already known, and it is specified by the element itself (for example, the element 2 is to be placed in position 2 in the ordered permutation). This additional information can be used to build different and more efficient sorting algorithms for permutations that do not resemble the classical ones.

3. Traditional sorting algorithms are designed to use the minimal number of moves (exchanges between elements), and also to do the minimal number of comparisons between elements. Crossover operators, like the traditional sorting algorithms, require sorting using the minimal number of moves. However, crossover operators do not require optimality in the number of comparisons.
4. The purpose of sorting algorithms is to sort the elements of a vector. The purpose of crossover operators is, instead, to return an offspring permutation on the minimal sorting trajectory between parent permutations. This can be done by interrupting the sorting algorithm at a random point and returning the partially sorted permutation as offspring.
5. There are two distinct categories of sorting algorithms: deterministic and non-deterministic. Deterministic sorting algorithms perform the sorting always in the same way given the same permutation to sort. For example, deterministic bubble sort scans the permutation always from left to right and applies the first adjacent swaps that make closer the current permutation to the complete ordered one. Non-deterministic sorting algorithms are randomised sorting algorithms that do not select the next move deterministically but according to some randomised strategy. For example, the uniform non-deterministic sorting strategy for bubble sort is scanning the current permutation considering all the sorting moves (those that get the current permutation one move closer to the complete order), and then selecting one at random and applying it. Crossover operators may benefit from non-deterministic sorting. That is, given the same permutation to sort, crossover may return offspring on different minimal sorting trajectories in different executions.

In the following two subsections we present two simple and efficient algorithms that implement geometric crossover and geometric mutation for permutations based on edit distances.

---

#### Algorithm 1 Sorting crossover

---

```

1: Input: par1, par2 Output: offsp
2: normalised_par1 = compose(par1, inverse(par2))
3: moves = sort(normalised_par1)
4: distance = length(moves)
5: xo-point = random_int([0, distance])
6: offsp = par1
7: While xo-point > 0 do
8:   offsp = compose(offsp, moves[xo-point])
9:   xo-point = xo-point - 1
10: end while
11: return offsp

```

---

### 6.2. Implementation of sorting crossovers

In Algorithm 1 we report the pseudo-code for a generic sorting crossover based on any edit move. Firstly, the elements of parent1 are renumbered (normalised) according to the order of the elements in parent 2. This is done by permutation composition of parent 1 with the inverse permutation of parent2. This transformation allows us to reduce the task of sorting one permutation toward a second arbitrary permutation into the standard task of sorting a permutation into a completely ordered permutation. The following step is obtaining the sequence of edit moves that order the normalised parent1 on a minimal sorting trajectory. This can be done by modifying any sorting algorithm that satisfies the requirement of being a “minimal sorting by  $x$ ” algorithm in a way that, while sorting, it collects the sequence of sorting moves it uses, and it returns it. The distance between the two parent permutations based on the edit move considered is the number of moves on the sorting trajectory between the two permutations returned by the modified sorting algorithm. A crossover point is then selected at random on the sorting trajectory and the offspring permutation is obtained by applying the sequence of moves to the first parent permutation until the crossover point has been reached.

We obtain two different types of crossover depending on whether we use a deterministic or a non-deterministic sorting algorithm as a basis for crossover. The offspring of a deterministic sorting crossover all lie on the same geodesic between their parents. The offspring of a non-deterministic sorting crossover, instead, cover all segment between their parents (because the union of all geodesics connecting two points coincides with the segment between them). Notice that the actual probability distribution of the offspring over the segment is not necessarily uniform, and it depends on the specific geometry of the space induced by the specific edit move considered.

**Algorithm 2** Mutation operator

---

```

1: Input: par, mutation prob  $p_m$  Output: offsp
2: offsp = par
3: While  $p_m > \text{uniform\_random}([0, 1])$  do
4:   neighbours = get_neighbours(offsp)
5:   offsp = uniform_random(neighbours)
6: end while
7: return offsp

```

---

## 6.3. Implementation of edit mutation

In Algorithm 2 we report a generic geometric mutation that can be used with any solution representation coming with a notion of edit distance (not only with permutations). The parameter  $p_m$  is the mutation probability. The neighbours of a given syntactic configuration are all those syntactic configurations within the reach of a single edit move. The mutation operator can reach any point in the space from any other with an exponentially decreasing probability for increasing distance.

## 7. Existing crossovers and permutation interpretations

There are a number of crossover operators defined for permutations.<sup>2</sup> Most of them were devised with a *specific interpretation* of the permutation in mind. This is reflected in their names. So, for example, Davis's *order crossover* [6] emphasises the fact that a permutation is seen as a relative order, *cycle crossover* [17] preserves absolute positions, and *edge recombination crossover* [22] focuses on the adjacency relation of the elements in the permutation.

Some crossovers achieve their goals of transmitting a specific relationship among elements from the parents to the children perfectly (*perfect crossovers*), others achieve their goals only approximately (*imperfect crossovers*). For example cycle crossover transmits perfectly the common positional information of parents to children and, so, it is a perfect crossover. Both Davis's order crossover and edge recombination are imperfect crossovers in that they are not able to transmit perfectly the common relative order of the parents and the adjacency relation, respectively. However, the common relative order is much easier to transmit perfectly than the adjacency relation (because the distance associated with the adjacency relation, reversal distance, is NP-Hard to compute). Indeed, another crossover, the merge

crossover [4], perfectly transmits the relative order of parents to children.

Some crossover operator is deliberately designed to be a trade-off, transmitting part of the relative order, part of the absolute position and part of the adjacency relation present in the parent permutations to the offspring permutations (*hybrid crossovers*). This is indeed possible since the three relations have subtle interdependencies. One of such crossover operators is the partially mapped crossover (PMX) [11]. Hybrid crossovers have the advantage to work reasonably well independently from the specific interpretation of the permutation. However, when hybrid crossovers are compared with perfect crossovers for a specific interpretation of the permutation on a problem in which this interpretation is relevant, the hybrid ones tend to perform worse than the perfect ones.

## 8. Geometricity of pre-existing crossovers

In this section, we consider some of the most frequently used recombination operators for permutations and analyse whether they are geometric crossovers and, if so, under what distances.

In section 6, we showed that geometric crossovers for permutations under edit distances are naturally associated with sorting algorithms. This allows us to implement these crossovers using sorting algorithms. However, using sorting algorithms is not the only way to implement these crossovers. Indeed, some pre-existing crossovers for permutations that fit the definition of geometric crossover under edit distances don't really look like sorting algorithms, even if in fact they are in fact related to them, as we will show briefly. Any crossover for permutations which is found to be geometric under edit distance produces offspring on the minimal sorting trajectory between parents according to some edit move. The reason why such a crossover may not look like a sorting crossover is because it picks offspring on a minimal sorting trajectory without generating explicitly the permutations on this trajectory. Instead, it generates directly offspring permutations by a single syntactic manipulation of the parent permutations equivalent to the application of a sequence of single edit moves on a minimal sorting trajectory between parents. This is made possible by the special property of permutations with regard to sorting (see section 6). The recombination operators for permutations analysed in the following, which are geometric crossovers, are sorting crossovers of the type just described.

<sup>2</sup> For a good overview see [1] and [2]. These describe the crossovers considered in this and the next section in detail.

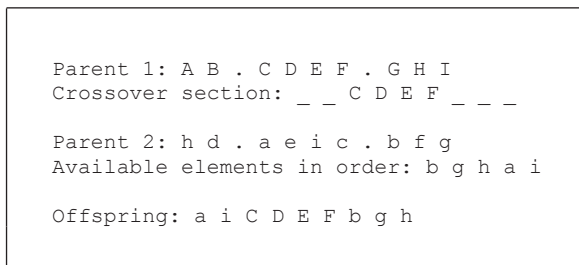


Fig. 1. Example of Davis's order crossover.

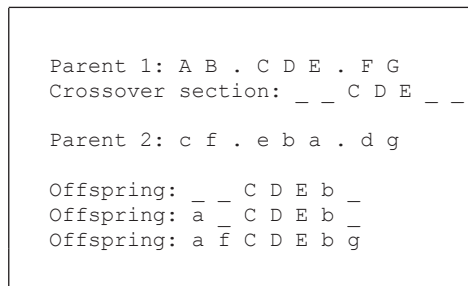


Fig. 2. Example of partially mapped crossover.

### 8.1. Order crossover

Davis's order crossover [6] was expressly designed to transmit information about the relative order. Figure 1 shows an example of this crossover. It begins by copying a randomly chosen segment of parent 1, the crossover section, into the offspring. Then, starting from the second crossover point in parent 2, it copies the remaining unused elements into the offspring in the order they appear in parent 2, wrapping around at the end of the list.

Davis's order crossover is a hybrid crossover that tends to preserve, only partially, all the three relations (adjacency, relative order and absolute position) among elements. We illustrate this in the following. The elements in the crossover section preserve relative order, absolute position and adjacency from parent 1. The elements copied from parent 2 do not preserve neither position nor adjacency. In the example in Fig. 1, position is not respected because, for example, at position two the offspring differs from both its parents. Adjacency is not respected either: in both parents  $B$  ( $b$ ) and  $C$  ( $c$ ) are adjacent, but they are not adjacent in the offspring. Also, the relative order information contained in the parents is passed to the offspring only partially. In the example,  $H$  ( $h$ ) precedes  $I$  ( $i$ ) in both parents, but in the offspring  $i$  precedes  $h$ . In [15], we proved that Davis's order crossover is not a geometric crossover.

### 8.2. Partially mapped crossover

Partially mapped crossover (PMX) was proposed by Goldberg and Lingle [11]. Figure 2 shows an example of this crossover.

Like order crossover, PMX begins by copying a randomly chosen segment of parent 1, the crossover section, into the offspring. Then, it considers the elements between the crossover points in parent 2 which

have not already been copied to the offspring. In the example, these elements are  $b$  and  $a$ . For each of those elements, it looks in the offspring to see what element has been copied in its place from parent 1. In the example,  $b$  in parent 2 is at the position of  $D$  in the offspring, and  $a$  in parent 2 is at the position of  $E$  in the offspring. Then PMX places each of these elements in parent 2, say  $b$ , in the offspring at the position occupied in parent 2 by the element which is in place of  $b$  in the offspring ( $D$ ). In the example, the position of  $D$  ( $d$ ) in parent 2 is immediately after the second crossover point, so  $b$  is placed at that position in the offspring. In the offspring, at the position of  $a$  in parent 2, there is  $E$ . The position in the offspring, corresponding with the position of  $E$  ( $e$ ) in parent 2, is already filled by another element ( $C$ ). In this case, the element coming from parent 2 ( $a$ ) goes in the offspring at the position occupied in parent 2 by the element  $C$  ( $c$ ), which in the example is position one. When finished with the elements of the crossover section, the rest of the offspring can be filled from parent 2 without changing positions of the elements. In the example,  $f$  and  $g$  are filled from parent 2 to the offspring.

PMX, like order crossover, is a hybrid crossover in that it tends to preserve, only partially, all the three relations (adjacency, relative order and absolute position) among elements. Adjacency is not perfectly transmitted: in the example, in both parents  $a$  ( $A$ ) and  $b$  ( $B$ ) are adjacent, but in the offspring they are not. Common relative order of the elements of the parents is not preserved either: in the example, in both parents  $b$  ( $B$ ) comes before  $d$  ( $D$ ), but in the offspring  $b$  comes after  $D$ . Also, position-wise values are not perfectly transmitted: in the example, at position six the elements in the parents are  $F$  and  $d$ , but the element at that position in the offspring ( $b$ ) is neither of them. Notice, however, that when both parents have the same element at the same position, also the offspring will have that ele-

Positions:	1	2	3	4	5	6	7	8	9	10	11	12
Parent 1:	A	B	C	D	E	F	G	H	I	J	K	L
Parent 2:	h	k	c	e	f	d	b	l	a	i	g	j
Cycle label:	1	2	3	4	4	4	2	1	1	1	2	1
Offspring:	A	k	C	e	f	d	b	H	I	J	g	L

Fig. 3. Example of cycle crossover.

ment at that position. In [15], we proved that PMX is geometric under swap distance.

### 8.3. Cycle crossover

Cycle crossover [17] is concerned with preserving as much information as possible about the absolute position in which elements occur. Figure 3 shows an example of this crossover. Cycle crossover has two phases. Firstly, it divides the elements into cycles. A cycle is a subset of elements that has the property that each element always occurs paired with another element of the same cycle when the two parents are aligned. Secondly, the offspring are created by selecting randomly cycles from the parents. In the example, the elements  $a, h, i, j$  and  $l$  belong to cycle 1, the elements  $b, g$  and  $k$  belong to cycle 2, the element  $c$  belong to cycle 3, and the remaining elements ( $d, e$  and  $f$ ) belong to cycle 4. The offspring was formed by passing to the offspring the elements of cycles 1 and 3 from parent 1, and the elements of cycle 2 and 4 from parent 2.

Since all elements in the offspring occupy the same positions as in one of the two parents, *cycle crossover preserves perfectly the absolute positions of the parents*. Relative order is not transmitted perfectly (see elements  $h$  and  $k$  in the example). Also, adjacency is not preserved perfectly (in the example, in both parents  $a$  and  $k$  are not adjacent, but in the offspring they are adjacent). In [15], we proved that Cycle crossover is geometric under Hamming distance restricted to permutations and that it is also geometric under swap distance.

### 8.4. Merge crossovers

Merge crossovers 1 and 2 [4] use a global precedence vector to recombine parent permutations (see Fig. 4). Given any two elements in the permutation, the global precedence vector indicates which element has higher priority for processing (elements which appear earlier in the vector have higher precedence).

Parent 1:	C	F	G	B	A	H	D	I	E	J
Parent 2:	E	B	G	J	D	I	C	A	F	H
Precedence:	A	B	C	D	E	F	G	H	I	J
Offspring:	C	B	G	F	A	H	D	E	I	J
Offspring:	C	E	B	F	G	A	H	D	I	J

Fig. 4. Example of merge crossover 1 and 2.

Let us consider merge crossover 1. Parents are processed from left to right. At position 1, parent 1 has  $C$  and parent 2 has  $E$ . Since  $C$  has precedence on  $E$ ,  $C$  is copied to the offspring at position 1. Since  $C$  has already been allocated a position in the offspring, the  $C$  which appears later in parent 2 is swapped with  $E$  at the position 1 of parent 2. The second position will be processed next in the same way. When all positions have been processed, both parents are transformed into copies of the same offspring. Merge crossover 2 differs from merge crossover 1 in that when an element is added to the offspring it is deleted from both parents instead of being swapped. The deletion treats permutations as lists. Therefore, the deletion of an element does not leave an empty position in the permutations. After all elements have been processed, both parents are transformed into empty lists.

Merge crossover 1 does not preserve perfectly the absolute position relation between elements (see elements of the parents and of the offspring at position 4 in the example). It does not transmit the adjacency relation either (elements  $F$  and  $A$  are non-adjacent in both parents, but they are adjacent in the offspring). Although in this example the common relative order of the elements of the parents is perfectly transmitted to the offspring, this does not hold in general.<sup>3</sup> In [15], we proved that Merge crossover 1 is a geometric crossover under swap distance.

Merge crossover 2 does not preserve perfectly the absolute position relation between elements (see elements of the parents and of the offspring at position 4 in the example). It does not transmit the adjacency relation either (elements  $B$  and  $G$  are adjacent in both parents, but they are not adjacent in the offspring). However, *merge crossover 2 transmits perfectly the*

<sup>3</sup> For a counterexample, consider parent permutations ( $ACDEB$ ) and ( $BACDE$ ), and precedence vector ( $BCDEA$ ). In this case, the offspring permutation is ( $BCDEA$ ). In both parents,  $A$  precedes  $C$ , but in the offspring  $C$  precedes  $A$ .

common relative order of the elements of the parents to the offspring. In [15], we proved that Merge crossover 2 transmits perfectly the common relative order of the elements of the parents to the offspring, and as a consequence it is a geometric crossover under ROD (Hamming distance between relative order matrices).

## 9. Sorting crossover for TSP

Edge recombination [22] is an operator expressly designed for TSP. It considers a solution as a tour of cities. Therefore, rather than being defined for permutations it is defined over *circular permutations*. In its various improvements its stated objective is to greedily recombine parent tours in order to transmit as much as possible the adjacency relation, introducing in the offspring tours the minimum number of “foreign” edges not present in either parent [22].

As in the linear case, also circular permutations can be represented with an adjacency matrix. Again, the segment between the parent circular permutations (under Hamming distance for the adjacency relation matrix) contains all the feasible offspring circular permutations that perfectly respect the adjacency relation of their parents. The geometric crossover for circular permutations under this notion of distance is well-defined and actually achieves what *edge recombination can only aspire to*. However, this is only a theoretical operator that cannot be directly implemented.

In the case of circular permutations, the block-reversal move is the notion of edit distance closest to the adjacency matrix distance. In a single application to a tour, this introduces the minimal change to the adjacency relation among elements in the permutation. This move is the well-known 2-change move, and it is the basis for successful local search algorithms for TSP [9]. Figure 5 shows the possible offspring (the segments) between two circular permutations (parents) under geometric crossover.

Analogously to the linear case, the circular permutations in the segment under reversal distance are those laying in a minimal sorting trajectory from a parent circular permutation to the other. Sorting circular permutations by reversals is NP-hard [20]. So, *the geometric crossover under this notion of distance cannot be implemented efficiently*.

Sorting circular permutations by reversals is tightly connected with the problem of sorting linear permutations by reversals. So, all the algorithms developed for the latter task can be used with minor modifications

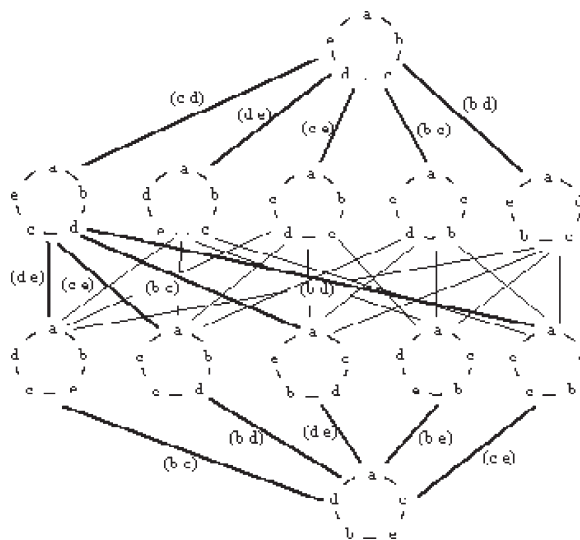


Fig. 5. Example of geometric crossover between two circular permutations. The parents are the configurations at the top and at the bottom. The set of all possible offspring are all intermediate configurations obtained by transforming (i.e., sorting) one parent into the other using the minimum number of reversal moves. As there are multiple minimal sorting trajectories, offspring configurations may lay on separate paths. The offspring are in the segment between the parents under reversal distance as, on minimal sorting trajectories, the sum of the reversal distances between any intermediate configuration and the initial and target configurations equals the reversal distance between those configurations.

also for the former [20]. Sorting linear permutations by reversals is NP-hard, too [5]. However, a number of approximation algorithms (running in polynomial time) exist to solve this problem within a bounded error from the optimum [14]. This allows implementing efficiently approximate geometric crossovers.

### 9.1. Experiments

We have implemented an efficient approximate geometric crossover for circular permutations under reversal distance based on the algorithm for sorting permutations by reversals by Kececioğlu and Sankoff [14].<sup>4</sup> This algorithm achieves a worst-case approximation of a factor of 2. In other words, in the worst case, it sorts permutations using twice the minimum number of reversals needed. However, on average the approximation factor is close to 1. It has (worst-case) time complexity  $O(n^2)$ , where  $n$  is the number of elements of the permutations to sort. The approximated

<sup>4</sup> The authors want to express their gratitude to John Kececioğlu and David Sankoff for making the source code of their algorithm available.

Table 1  
Results of the comparison of recombination operators for the TSP

Problem(Opt)	PMX				ERX				SBRX			
	Fitness		Iterations		Fitness		Iterations		Fitness		Iterations	
	avg	std	avg	std	avg	std	avg	std	avg	std	avg	std
eil51(426)	631.4	38.6	81369.7	10036.5	426.4	1.2	118221.3	11596.1	431.8	4.6	84594.2	5340.6
gr96(51229)	149118	12110.8	210688	27685.9	58608.7	5356.5	427072	100460.5	61446.7	1633.2	258560	12116.1
eil101(629)	1413.4	86.2	213042.7	19913.4	738.1	63.3	369458	127840.5	686.4	27	272498	18398.3
lin105(14379)	40969.6	3257.5	247520	23564.5	16515.1	1675.4	481460	148344.8	15804.9	368.5	312900	15602.3
d198(15780)	54746.1	4052.8	750684	63448	29673.5	1430	829224	187930	18609.8	542.1	908292	129387.5
kroA200(29368)	141416.3	11771.9	634666.7	85717.9	58603.4	1760.4	763200	95640.4	38282.9	836.7	873200	59653.7
lin318	271538.3	12078.8	1284296	100264	113785.3	6947.5	1341960	324625.9	76735.3	5069.9	1833376	104026.1
pcb442	354119.5	14325.1	2110403	155927.4	166380.4	11564.3	2079463	435060.2	106840.9	2775.2	3023575	193856

geometric crossover uses the algorithm for sorting permutations by reversals to generate all permutations on a sorting trajectory by reversals between the two parent permutations, and it returns one of those permutations drawn at random as offspring.

We have tested our sorting crossover on the following frequently used problem instances of the well-known library of TSP problems TSPLIB<sup>5</sup>: eil51, gr96, eil101, lin105, d198, kroA200, lin318 and pcb442. The number in the name is the size of the instance (number of cities).

We compared three different recombination operators under the same setting: PMX, Edge Recombination (EX), which is known to be a very good operator for TSP, and sorting by reversal geometric crossover (SBRX). Since we wanted to compare the performance of the recombination operators alone, no mutation was used. We used a generational genetic algorithm with probability of crossover 1 and linear ranking. The population size was set to 20 times the problem instance size. As stop criterion we used population convergence. The ratio between population size and problem instance was chosen to allow the best recombination to reach a near optimal solution before population convergence for all instances. No other parameter tuning was performed.

We run 30 independent runs for each problem. In Table 1 we report, for each operator, the average and the standard deviation of the best fitness found, and the average and standard deviation of the number of fitness evaluation until convergence. In terms of solution quality, PMX is always the worst. ERX performs slightly better than SBRX for small instances (eil51 and gr96), but SBRX performs better on all the other instances. In particular, SBRX performs better and bet-

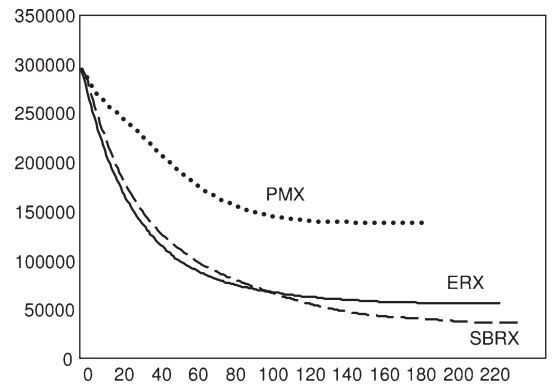


Fig. 6. A typical run (on instance KroA200) showing the performance curves for PMX (dotted line), ERX (solid line), and SBRX (dashed line). Abscissas represent generations while ordinates report the average fitness over 30 runs of the best individual in the population.

ter against ERX as the instance size grows. In terms of number of iterations to convergence, better performing operators tend to take more time to converge. However, the extra time is only a fraction of the overall running time.

In Fig. 6 we show a plot of the fitness of the best individual in the population over time for a run with the problem KroA200. This is a typical run: PMX produces the worst result in terms of fitness and stops earlier than the other two recombination operators; ERX has better fitness than SBRX until halfway, then SBRX continues to improve and overtaking ERX. ERX converges earlier than SBRX.

## 10. Conclusions

Permutations differ strongly from binary strings and real vectors, from which the geometric framework has originally arisen. In particular the geometric notion of orthogonality, common to binary strings and real vec-

<sup>5</sup> TSPLIB can be downloaded from <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>.

tors, is not present in permutations. Given this, one might wonder about the meaningfulness on applying a geometric framework to permutations.

As we have shown in this paper, however, not only the geometric framework can be applied to permutations, but, perhaps more importantly, it sheds light on this representation, revealing a hidden and intimate connection between geometry of permutations, crossover and sorting algorithms.

Geometric crossover and mutation are well-defined once one has a notion of distance over the solution set. The permutation representation allows for three notions of non-edit distances connected with the permutation interpretations. Three geometric crossovers based on the permutation interpretations are therefore well-defined. However, such geometric crossovers are hard or even impossible to implement efficiently, in that they are based on non-edit distances.

Most of the pre-existing crossover operators for permutations are designed around interpretations. We have shown that they fit, some exactly and some imperfectly, the geometric crossover definitions connected with permutations interpretations. The permutation representation also allows for a number of edit distances connected with various notions of mutation. Each notion of edit distance induces a notion of geometric crossover. Because of the distance duality, under a given edit distance a point on a segment between two permutations is on a minimal sorting trajectory connecting the two permutations. This allows implementing such crossovers using *sorting algorithms*. Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Other edit distances give rise to crossovers can be implemented efficiently (in polynomial time) only in an approximated way.

We have shown that a number of important recombination operators for permutations used in practice are geometric crossovers under different edit distances: PMX is geometric under swap distance, cycle crossover is geometric under both hamming distance restricted to permutations and swap distance, merge crossovers 1 and 2 are geometric crossovers under swap distance and relative order distance, respectively. Also, edge recombination aims to be a geometric crossover under the distance associated with the adjacency relation among elements of the permutation, but it succeeds only partially in this.

Circular permutations are tightly connected to traditional permutations but they do not coincide. We have shown how to apply geometric crossover to TSP that is naturally defined over circular permutations. The fit-

ness landscape for the TSP problem based on reversals distance for circular permutations is smoother than with any other distances; for this reason one might expect the associated geometric crossover to perform well. Experimental results agree with this prediction: our proposed geometric crossover for TSP outperforms edge recombination, which is one of the best crossover for this problem.

## References

- [1] T. Bäck, D.B. Fogel and T. Michalewicz, eds. *Evolutionary Computation 1: Basic Algorithms and Operators*, Institute of Physics Publishing, 2000.
- [2] T. Back, D.B. Fogel and Z. Michalewicz, eds. *Handbook of Evolutionary Computation*, Oxford Press, 1997.
- [3] T. Bäck and H.-P. Schwefel, An overview of evolutionary algorithms for parameter optimization, *Evolutionary Computation* 1(1) (1993), 1–23.
- [4] J. Blanton and R. Wainwright, Multiple vehicle routing with time and capacity constraints using genetic algorithms, in: *Proceedings of the International Conference on Genetic Algorithms*, 1993, pp. 452–459.
- [5] A. Caprara, Sorting by reversals is difficult, in: *Proceedings of the 1st Annual International Conference on Computational Molecular Biology*, 1997, pp. 75–83.
- [6] L. Davis, Applying adaptive algorithms to epistatic domains, in: *Proceedings of the International Joint Conference on Artificial Intelligence*, 1985, pp. 162–164.
- [7] M. Deza and T. Huang, Metrics on permutations, a survey. *Journal of Combinatorics, Information and System Sciences* 23 (1998), 173–185.
- [8] B.R. Fox and M. McMahon, Genetic operators for sequencing problems, in: *Proceedings of the Workshop on the Foundations of Genetic Algorithms*, 1991, pp. 284–300.
- [9] F. W. Glover, ed, *Handbook of Metaheuristics*, Kluwer, 2002.
- [10] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, 1989.
- [11] D.E. Goldberg and R. Lingle, Alleles, loci, and the traveling salesman problem, in: *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, 1985, pp. 154–159.
- [12] S. Gustafson and L. Vanneschi, Operator-based tree distance in genetic programming, *IEEE Transactions on Evolutionary Computation* 12(4) (2008), 506–524.
- [13] J.H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [14] J. Kececioğlu and D. Sankoff, Exact and approximate algorithms for sorting by reversals, with applications to genome rearrangement. *Algorithmica* 13 (1995), 180–210.
- [15] A. Moraglio, *Towards a Geometric Unification of Evolutionary Algorithms*, PhD Thesis, University of Essex, 2007.
- [16] A. Moraglio and R. Poli, Topological interpretation of crossover, in: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004, pp. 1377–1388.
- [17] I. Oliver, D. Smith and J. Holland, A study of permutation crossover operators on the traveling salesman problem, in: *Proceedings of the Second International Conference on Genetic Algorithms* 1987, pp. 224–230.

- [18] R. Poli, W.B. Langdon and N.F. McPhee, *A field guide to genetic programming*. Published via <http://lulu.com> and freely available at <http://www.gp-field-guide.org.uk>, 2008, (With contributions by J.R. Koza).
- [19] F. Rothlauf, *Representations for Genetic and Evolutionary Algorithms*, Springer, 2002.
- [20] A. Solomon, *Sorting by Bounded Permutations*, PhD Thesis, Virginia Polytechnic Institute, 1997.
- [21] J.P.C. Vergara, *Sorting by Bounded Permutations*, PhD Thesis, Virginia Polytechnic Institute, 1997.
- [22] D. Whitley, T. Starkweather and D. Shaner, Travelling salesman and sequence scheduling: quality solutions using genetic edge recombination, in: *Handbook of Genetic Algorithms*, L. Davis, ed, Van Nostrand Reinhold, 1991, pp. 350–372.