# Abstract Geometric Crossover for the Permutation Representation

Alberto Moraglio and Riccardo Poli

Department of Computer Science, University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, UK
{amoragn,rpoli}@essex.ac.uk

**Abstract.** Abstract crossover and abstract mutation are *representation-independent* operators that are well-defined once a notion of distance over the solution space is defined. They were obtained as generalization of genetic operators for binary strings and real vectors. In this paper we explore how the abstract geometric framework applies to the permutation representation. This representation is challenging for various reasons: because of the inherent difference between permutations and the representations that inspired the abstraction; because the whole notion of geometry over permutation spaces radically departs from traditional geometries and it is almost unexplored mathematical territory; because the many notions of distance available and their subtle interconnections make it hard to see the right distance to use, if any; because the various available interpretations of permutations make ambiguous what a permutation represents, hence, how to treat it; because of the existence of various permutation-like representations that are incorrectly confused with permutations; and finally because of the existence of many mutation and recombination operators and their many variations for the same representation. This article shows that the application of our geometric framework naturally clarifies and unifies an important domain, the permutation representation and the related operators, in which there was little or no hope to find order. In addition the abstract geometric framework is used to improve the design of crossover operators for well-known problems naturally connected with the permutation representation.

Technical Report CSM-429 ISSN 1744-8050

## 1. Introduction

In previous work (Moraglio & Poli, 2004) we introduced a representation-independent geometric generalization of genetic operators for binary string representation and real vector representation. The geometric definitions of mutation and crossover introduced are based on the distance associated with the search space, seen as a metric space, and on the simple geometric notions of ball and line segment. This way of connecting genetic operators and fitness landscape is opposite of the standard approach where the fitness landscape is a function of the genetic operators, hence leading to the one operator-one landscape paradigm (Jones, 1995). Seeing genetic operators as functions of the search space produces a great deal of simplification and clarification: mutation and crossover share the same simple search space, that naturally corresponds to the classical notion of neighbourhood structure used by many meta-heuristics, and their relationship becomes clear.

Since the notion of landscape and abstract genetic operators are of central importance in our framework, we will present them in this article in sections 2.1 and 2.2, where we recast them in a more abstract and general way emphasising the geometric interpretation of the search space. This framework allows treating genetic operators for discrete search spaces and continuous search spaces in a uniform way, since at an abstract level, both types of spaces come with a notion of distance, and is thus all is needed to fully-define our genetic operators.

In this framework the definitions of genetic operators are representation-independent but they can be, through a dual interpretation of the notion of distance of the search space, tightly rooted in the syntax of the specific representation considered. This has surprising and important consequences including the unification in a single framework of all evolutionary algorithms based on different solution representations, a general unitary theory of evolutionary algorithms and principled crossover design for any representation (Moraglio & Poli, 2004). Section 2.3 is devoted to introducing duality and clarifying its origin.

Since our definitions of genetic operators are generic and are connected neither with the solution representation nor with problem at hand, it is important to understand how they relate with the NFL theorem (Wolpert & Macready, 1996). In particular we need to explain how problem knowledge can be specified and used by the formal evolutionary algorithm to perform better than random search. The key is the difference between problem and landscape, the former being given and the latter being designed. We will see, in section 2.4, how the landscape is a knowledge interface between formal algorithm and formal problem and how through a domain-specific solution representation and a move operator that makes sense for the problem, one can easily and naturally design such a landscape.

In section 2, many aspects of the framework are discussed only abstractly; in the rest of the article these concepts are put at work on concrete examples. Since the notion of distance is so fundamental for the application of the geometric framework, in section 3 we analyse various notions of distance for permutations, their origins, their interdependencies and their use within the framework. Section 4 is devoted to the important special case of edit distances for permutations and the related mutation operators.

The major surprise brought about by the application of the general geometric definition of crossover to the permutation representation consists in the discovery that the notion of crossover for permutation is intimately connected with the notion of sorting algorithm. This is ultimately due to the fact that doing geometry using distances that are firmly rooted in the syntax of a specific representation, such as edit distances, allows for a dual interpretation of geometric objects and transformations. For permutations, the geometric notion of shortest path or geodesic, that is important in our framework because a segment is the union of all geodesics between two points, corresponds dually to the syntactic notion of minimal sorting trajectory. This creates the dual connection between abstract crossover and sorting algorithms mentioned above. In section 5, we explore this connection thoroughly clarifying what the similarity and differences between crossovers and ordinary sort algorithms are. In section 6, we propose a set of new crossover operators based on well-known sorting algorithms.

In addition, in section 7, we show that pre-existing crossover operators for permutations match the abstract definition of crossover. So, even if they do not look like sorting algorithms, they fit the geometric definition of crossover and so they are sorting algorithms in disguise!

One peculiarity of the permutation representation that we will encounter is that there are many equivalently natural notions of crossover; hence one may wonder what the

right notion of crossover is after all. For the NFL theorem (Wolpert & Macready, 1996) there cannot be the right crossover: what is a good crossover for a problem must be necessarily bad for another one. The question is therefore how to match crossover and problem in a principled way. In section 8 we discuss three heuristic methods to do that: build a crossover using a good mutation, build a crossover using a neighbourhood based on the small-move small-fitness change design principle, or build a crossover using a distance that is connected with a distance that is relevant for the solution interpretation.

In section 9, we propose the n-queens problem. We do an experimental comparison of a number of crossovers based on three classical sorting algorithms: selection sort, bubble sort and insertion sort.

Permutations, circular permutations and permutations with repetitions are generally confused. Even if they are indeed related solution representations, they are not the same and, in particular, they give rise to related but different notions of edit distance and, consequently, different but related notions of crossover. In section 10, we discuss the case of circular permutations in connection with an important problem that is naturally based on such a representation, the travelling salesman problem (TSP).

In the context of devising a crossover for TSP or for crossover based on bounded edit moves we encounter a new aspect of distance duality and crossover implementation: the computational barrier. Only crossover defined using distances that use simple syntactic moves allow for an efficient implementation. When the distance is not tightly connected with the solution representation, the duality geometry/syntax is not enough to guarantee a straightforward implementation of geometric crossover by simple syntax manipulation of the solution representation. Therefore even if there is no reason in principle to define crossover over a distance defined via a solution representation there is a good computational reason to do so! We discuss this for the TSP problem in section 10.

In (Moraglio, 2000) was introduced a natural and effective crossover for JSSP. In section 11 we show that such a crossover fits perfectly the definition of abstract geometric crossover. This is another piece of evidence that the abstract geometric definition of crossover captures the general notion of crossover. In this case we did not use a crossover defined over the solution representation space directly (schedule) but we encoded schedules as permutations. However we show that, since the encoding is *betweenness preserving*, the crossover on permutations corresponds to a crossover over schedules in their original domain. Finally, in section 12, we present our conclusions.

## 2. Geometric framework

In previous work (Moraglio & Poli, 2004) we have given representation-independent definitions of crossover and mutation operators linked tightly to the notion of fitness landscape. We named these abstract operators topological operators because they are defined over the connectivity structure of the underlying search space. However, since they admit an important geometric interpretation, we prefer now to call them *abstract geometric operators* to stress the fact that we treat the search space as a geometric space.

In the original framework, we defined the distance in the search space starting from the syntax of solutions. In section 2.1 we give some necessary preliminary definitions. In section 2.2 we recast the original representation-independent definitions of genetic operators in a more abstract form, separating completely the notion of distance with its origin in the solution representation. We embrace an axiomatic approach and accept as a distance any function that meets the metric axioms, whether originating from syntax, real vectors or something else. In section 2.3, we analyse then the consequences of specifying the distance in the search space through the solution representation and introduce the important notion of duality that arises when doing geometry using a notion of edit distance. We will show that duality has surprising consequences. Finally, in section 2.4, we connect formal problem and solution representation, clarifying that ultimately is this connection that embeds problem knowledge into an evolutionary algorithm.

### *2.1 Preliminary definitions*

#### *2.1.1 Formal search problem*

Let *S* denote the *solution set*[1] comprising all the candidate solutions to a given *search problem P*. Here we distinguish between the concept of solution and its representation. The solutions in the solution set must be seen as *formal solutions* not relaying on any specific underlying representation. The goal of a search problem *P* is to find specific solution/s in the search space that maximize (minimize) an *objective function* $g : S \to R$. We write *P=(S, g)*. Notice that global optima are well defined when the objective function is well defined. In particular, they are independent of any structure defined on *S*. On the contrary, *local optima* are defined only when a structure over *S* is defined. It is very important to understand that *a search problem in itself does not come with any predefined structure or notion of distance over the solution set*.

#### *2.1.2 Formal fitness landscape*

Formally, a *metric space* (*M*, *d*) is a set *M* provided with a metric or distance *d* that is a real-valued map on $M \times M$ which fulfils the following axioms for all $s_1, s_2 \in M$ :

1. $d(s_1, s_2) \geq 0$ and $d(s_1, s_2) = 0$ if and only if $s_1 = s_2$ ;
2. $d(s_1, s_2) = d(s_2, s_1)$, i.e. *d* is symmetric; and
3. $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$, i.e. *d* satisfies the triangle inequality.

We assign a structure to the solution set by endowing it with a notion of distance *d* respecting the axioms for a metric. *M=(S, d)* is therefore a *solution space* and *L=(M, g)* is the corresponding fitness landscape. *Notice that d is an arbitrary distance and need not have any particular connection or affinity with the search problem at hand.* Once we have a landscape, local optima are well defined. *The same solution set can be associated with more than one distance giving rise to different search spaces for the same search problem.* The distinction between search problem and fitness landscape is crucial to understand how problem knowledge is embedded in the search (see Section 2.1.4).

[1] We distinguish between *solution set* and *solution space*. The first refers to a collection of elements, while the second implies a structure over the elements.

*2.1.3 Metric geometry*

In classical Euclidean geometry, the measure of the distance between two points in the plane, say $A$ and $B$, is calculated using the well known formula: $d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$ . This is certainly a very intuitive notion of distance. By redefining the distance function between two points one obtains a new geometry for each distance redefinition. One simple example is the so-called 1st order Minkowski distance: $d(A,B) = |x_A - x_B| + |y_A - y_B|$. This definition of distance is fairly natural: it is the minimum distance that a taxicab would need to travel to reach point $B$ from point $A$, if all streets are only oriented vertically and horizontally. For this reason, this metric is often referred to as the Manhattan metric. Many geometric figures, like circles, ellipses, parabolas, are defined in terms of distance. For instance, a circle is just the set of points with a fixed distance to the centre. These, of course, look quite different if we use a non-Euclidean measure of distance.

If we go further and say that a shape corresponds to a particular definition independently from the specific notion of metric used, we are then dealing with *abstract shapes* that are defined axiomatically and present abstract geometric properties that are shape-specific but not distance-specific. These abstract shapes are studied in *metric geometry*. Two of them, balls and segments, turn out to be very useful to define abstractly mutation and crossover and in the following we consider them in more detail.

*2.1.4 Balls and segments*

In a metric space $(S,d)$ a *closed ball* is the set of the form $B(x;y) = \{ y \in S \mid d(x,y) \le r \}$ where $x \in S$ and $r$ is a positive real number called the radius of the ball. A *line segment* (or closed interval) is the set of the form $[x;y] = \{ z \in S \mid d(x,z) + d(z,y) = d(x,y) \}$ where $x, y \in S$ are called extremes of the segment. Note that $[x;y] = [y;x]$. The length $l$ of the segment $[x;y]$ is the distance between a pair of extremes $l([x;y]) = d(x,y)$. Let $H$ be a segment and $x \in H$ is an extreme of $H$, there exists only one point $y \in H$ , its conjugate extreme, such that $[x;y] = H$ . Examples of balls and segments for different spaces are shown in Figure 1. Note how the same set can have different geometries (see Euclidean and Manhattan spaces) and how segments can have more than a pair of extremes. E.g. in the Hamming space, a segment coincides with a hypercube and the number of extremes varies with the length of the segment, while in the Manhattan space, a segment is a rectangle and it has two pairs of extremes. Also, a segment is not necessarily "slim": it may include points that are not on its boundaries. Finally, a segment does not coincide with a shortest path connecting its extremes (*geodesic*). In general, there may be more than one geodesic connecting two extremes.

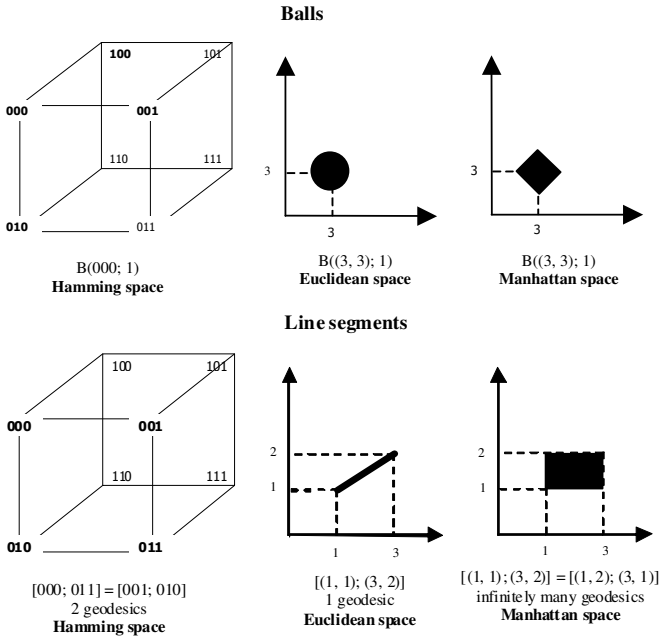


Fig. 1. Balls and segments for different spaces

### *2.2 Abstract geometric operators and formal evolutionary algorithm*

*2.2.1 Formal genetic operators*

We *define* two *classes* of operators in the landscape (i.e. using the notion of *distance* coming with the landscape): abstract mutation and abstract crossover. Within these classes, we identify two *specific* operators: abstract uniform mutation and abstract uniform crossover.

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

$$\Pr\{OP(p_1, p_2, ...p_g) = c\} = \Pr\{OP = c \mid P_1 = p_1, P_2 = p_2, ..., P_g = p_g\} = f_{OP}(c \mid p_1, p_2, ...p_g)$$

*Mutation* is a unary operator while *crossover* is typically a binary operator.

**Definition 1** *The* image set *or accessibility of a genetic operator OP is the set of all possible offspring produced by OP with non-zero probability when the parents are* $p_1, p_2, ...p_g$*:* $\mathrm{Im}[OP(p_1, p_2, ...p_g)] = \{c \in S \mid f_{OP}(c \mid p_1, p_2, ...p_g) > 0\}$

Notice that the image set is a *mapping* from a vector of parents to a set of offspring.

**Definition 2** *A unary operator M is a abstract ε-mutation operator if* $\mathrm{Im}[M(p)] \subseteq B(p; \varepsilon)$ *where ε is the smallest real for which this condition holds true.*

In other words, in *a abstract ε-mutation* all offspring are at most *ε away* from their parent.

**Definition 3** *A binary operator CX is a abstract crossover if* $\mathrm{Im}[CX(p_1, p_2)] \subseteq [p_1; p_2]$.

This simply means that in a abstract crossover offspring lay *between* parents. We use the term *recombination* as a synonym of any binary genetic operator.

We now introduce two *specific* operators belonging to the *families* defined above.

**Definition 4** *Abstract uniform ε-mutation UM is a abstract ε-mutation where all z at most ε away from parent x have the same probability of being the offspring:*

$$f_{UM\varepsilon}(z \mid x) = \Pr\{UM = z \mid P = x\} = \frac{\delta(z \in B(x,\varepsilon))}{|B(x,\varepsilon)|}$$

$$\mathrm{Im}[UM_\varepsilon(x)] = \{z \in S \mid f_{M\varepsilon}(z \mid x) > 0\} = B(x,\varepsilon)$$

*where* $\delta$ *is a function which returns 1 if the argument is true , 0 otherwise.*

When ε is not specified, we mean ε = 1.

**Definition 5** *Abstract uniform crossover UX is an abstract crossover where all z laying between parents x and y have the same probability of being the offspring:*

$$f_{UX}(z \mid x, y) = \Pr\{UX = z \mid P1 = x, P2 = y\} = \frac{\delta(z \in [x, y])}{|[x, y]|}$$

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

*These definitions are representation-independent and therefore the operators are well-defined for any representation.*

*2.2.2 Formal evolutionary algorithm*

Many evolutionary algorithms look alike when cleared from algorithmically irrelevant differences, such as authorship, historical origin, domain of application, phenotype interpretation and representation-independent algorithmic characteristics that, in effect can be freely exchanged between algorithms such as, for example, the selection scheme. Ultimately, the origin of the differences of the various flavours of evolutionary algorithms is rooted in the *solution representation* and relative *genetic operators*.

Since we have been able to fully define genetic operators in a complete abstract way without any reference to the solution representation we can now define rigorously formal evolutionary algorithms around the definitions of formal genetic operators. A general, formal evolutionary algorithm[2] is shown in Figure 2.

[2] A different notion of formal evolutionary algorithm, based on abstract genetic operators that are not based on fitness landscape, was introduced in (Surry & Radcliffe, 1996).

**FORMAL EVOLUTIONARY ALGORITHM(d)**
*Generate* (P(0))
$t \leftarrow 0$
**WHILE NOT** *Termination_Criterion* (P($t$)) **DO**
    *Evaluate* (P($t$))
    P' ($t$) ←*Select* (P($t$))
    P''($t$) ←*Apply_Abstract_Operators* (P'($t$),d)
    P($t$+1) ←*Replace* (P($t$), P''($t$))
    $t \leftarrow t + 1$
**END**
**RETURN** *Best_Solution*

Figure 2- Formal evolutionary algorithm

It is important to notice that the whole formal evolutionary algorithm is a function of the distance $d$ of the search space. Since such a distance can be chosen arbitrarily independently from the problem at hand, the formal evolutionary algorithm is fully-defined yet completely disconnected from the problem. *Naturally, NFL holds over all the possible choices of distance functions over the solution set.* To get better performance than random search for *any problem* is necessary to match problem and formal algorithm by choosing a distance that helps the search.

### *2.3 Specifying distance via solution representation*

In section 2.2 we have given abstract definitions of genetic operators that do not rely on any specific distance and that are completely disconnected from solution representation. In this section we will link solution representation to genetic operators by considering the application of abstract operators to a class of distances rooted in the solution representation: edit distances. This has a number of surprising consequences.

#### *2.3.1 Configuration space and edit distance*

A *configuration space* $C$ is a pair ($G$, *Nhd*) where $G$ is a set of syntactic configurations (syntactic objects or genotypes) and $Nhd : G \to 2^G$ is a syntactic neighbourhood function which maps every configuration in $G$ to the set of all the configurations in $G$ which can be obtained by applying a unitary syntactic modification operator (edit move). The edit move must be reversible (i.e. $y \in Nhd(x) \Leftrightarrow x \in Nhd(y)$) and connected (any configuration can be transformed into any other by applying a finite number of edit moves). Notice that given the same configuration set $G$ there may be more than one configuration space sharing the same configuration set but differing in the syntactic neighbourhood function (using a different edit move). A configuration space $C$=($G$, *Nhd*) is therefore endowed with a *syntactic neighbourhood structure* induced by the syntax of the configurations and the particular notion of edit move adopted. Such a neighbourhood structure can be associated with an undirected *syntactic neighbourhood graph* where vertices represent configurations and edges represent the relation of neighbourhood between configurations. The function returning the length of the shortest path (number of edges) connecting each pair of vertices in this graph respects all the axioms of a metric and is called *edit distance*. From a syntactic point of view, the edit distance

between two configurations correspond to the minimum number of edit moves needed to transform one configuration into the other. Notice that the same configuration set can be associated to different edit moves giving rise to different notion of edit distance for the same set.

*2.3.2 Connecting representation, mutation and crossover*

In (Moraglio & Poli, 2004) we provided two results that clarify the connection between representation, mutation and crossover. The first result is that given any notion of abstract $\varepsilon$-mutation there is a corresponding notion of abstract crossover and *vice versa*. Hence, there is one-to-one connection between mutation and crossover. The second result is that each solution representation may admit more than one notion of crossover and corresponding mutation: one for each edit distance applicable to that representation.

*2.3.3 Distance duality: edit distance geometry*

Descartes (Descartes, 1637) introduced the idea of *duality* in geometry by drawing a one-to-one correspondence between (vectors of) real numbers to points of the Euclidean space through the notion of coordinates. This duality taught at school to everyone, the Cartesian coordinate system, seems so obvious and natural nowadays that its importance is somehow invisible to the most. The remarkable thing of the Cartesian duality is that it connects two different worlds, numbers and geometry, and allows manipulating algebraically geometric shapes. This duality is the basis of analytic geometry and its successor, calculus. Here we introduce a generalization of Cartesian duality drawing a correspondence between syntactic objects and points of non-Euclidean discrete geometric spaces. We refer to these geometries as *edit distance geometries*.

The edit distance is a measure of syntactic dissimilarity between two syntactic objects. Every syntactic object uniquely represents a point in the space induced by the syntax and the notion of edit move considered. The induced space naturally inherits the edit distance from the syntax. However, in the transition the distance loses its meaning of dissimilarity measure and becomes a measure of spatial remoteness between points of the space. When one uses this distance as base for metric geometry, one actually uses a measure of syntactic similarity as if it were a notion of spatial remoteness. This creates the dual bond between syntax and space. So, any geometric definition based on edit distance has also a syntactic meaning. For example, the definition of a ball of radius one corresponds to all points that are in space within the radius (geometric interpretation) and also correspond to all syntactically related configurations that differ in their syntax of an edit unit from a given one (syntactic interpretation). So, geometric shapes can be interpreted as syntactic templates. This natural duality of edit distance geometries is important because allows to use instruments and ideas from geometry to reason about syntactic objects and vice versa. We call this property *distance duality*. Distance duality is a never-ending source of surprises because for each choice of syntax and edit move the syntactic interpretation of the same geometric definition is different and peculiar of the specific syntax and move considered. This is the same type of surprise that arises when one draws a ball using alternative notions of distance for real vectors, say Euclidean and Manhattan, obtaining respectively a rounded-ball and a diamond-ball. This has powerful consequences for evolutionary algorithms. One important consequence of duality is that the geometric definition of crossover, being also a syntactic definition, tells exactly what crossover is for any

choice of syntax and edit move. In Section 5, we show this for geometric spaces based on edit distances for permutations: the geometric notion of moving along a geodesic between points *a* and *b* (shortest path) corresponds syntactically to sorting the permutation that represent point *a* toward the permutation that represent point *b* on the minimal sorting trajectory. As a consequence, abstract geometric crossover for permutations can be implemented by a minor modification of traditional sorting algorithms.

### *2.4 Representation mapping and natural solution representation*

#### *2.4.1 Fitness landscape as knowledge interface*

How does the generic nature of a formal evolutionary algorithm relate to the NFL theorem and problem knowledge? To understand where the problem knowledge used by a formal evolutionary algorithm resides, we must make a clear distinction between problem and landscape. The problem is something *given* and it does not come with a predefined structure over the solution set. A landscape is a problem plus a structure over the solution set and it is something that is *designed*. Abstract genetic operators are defined over the connectivity structure of the landscape through a notion of distance arising from it; hence the way the landscape is designed ultimately determines the performance of the operators for a given problem. An abstract operator in itself has no knowledge of the problem and when matched blindly with the problem, for the NFL, its expected performance equals random search (Radcliffe & Surry, 1995). The problem knowledge is therefore embedded in the way the connectivity of the search space is mapped to the solution of the problem, that is to say that *the fitness landscape plays the role of a knowledge interface between formal problem and formal evolutionary algorithm.*

#### *2.4.2 Connection between formal problem and solution representation*

We have a formal solution set S and a configuration space C=(G,d). The genetic operators are implemented using the configuration space C; in this space the genetic operators are fully defined (since the space comes with a distance) and implementable in practice (since the space is defined over a syntax). As we mentioned earlier we treat S as a formal set of solutions and in particular we don't assume any specific solution representation. In order to connect the problem to the algorithm we need to define a representation mapping $r: S \to C$ that maps every solution in S to a configuration in C. For simplicity we consider *r* to be bijective. Now we can use the notion of distance *d* defined over the configuration space as a distance between formal solutions through $r^{-1}$. Notice that if *r* is an arbitrary mapping then the problem at hand and the evolutionary algorithm are not matched; hence for the NFL the expected performance of the evolutionary algorithm is the same as random search.

#### *2.4.3 Natural fitness landscape*

To match problem and algorithm in order to get expected performance better than random search the representation mapping *r* has to be chosen in a way to put some problem knowledge in the evolutionary algorithm. For many meta-heuristics based on neighbourhood search, in practice this is done quite straightforwardly in a heuristic way. The neighbourhood structure is built using a *domain-specific representation*, plus a neighbourhood function derived by *studying the objective function of the problem*, so that a syntactic unitary move corresponds to a little variation in the value of the objective function. For example, for the TSP the domain-specific representation

would be a city tour and the move 2-opt, inverting a sub-path of the tour, is chosen heuristically motivated by the fact that looking into the objective function is the one that changes the minimum number of weights contributing to changing the fitness of two neighbouring solutions. The theoretical reason behind this heuristic use of problem knowledge is not yet understood, but it is well-known that this is a good way to build the search space for most meta-heuristics (Pardalos & Resende, 2002).

## 3. Distances between permutations

In the previous sections we reported and substantially extended the geometric framework for evolutionary algorithm introduced in (Moraglio & Poli, 2004). The framework arose as a representation-independent generalization of crossover for binary strings and real-coded GAs but it is completely general. In this section we apply the framework to the permutation representation.

Differently from binary strings, where a single, natural definition of distance, the Hamming distance, is universally accepted, for permutations many notions of distance are equally natural. Such distances relate to each others in various ways with subtle dependencies. Further complication arises from the fact that permutations and circular permutations (and also permutations with repetitions to a lesser extent) are treated as if they were the same representation, which is incorrect: they are different representations and they allow for different notions of distance (for a survey on metrics on permutations see (Deza & Huang, 1998)).

Since the notion of distance is so fundamental for the application of the geometric framework, in the following we analyse various notions of distance for permutations, their origins, their interdependencies and their use within the framework. 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 (Back et al, 2000). 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 in this case 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 (C D E B F A). If the adjacency is important then the fact that the elements D and E are adjacent is relevant as well as the fact that the elements C and B are not adjacent. If the important aspect is the relative order then what is relevant is the fact that D precedes E and that C precedes B. If the absolute order is important then the relevant point is that C is in position 1, D in position 2, etc.

For each interpretation of permutation, it is possible to write a binary matrix that represents the actual relation among elements in the permutation. So, we can have a relative order matrix, an absolute position matrix and an adjacency matrix. It is possible to define three distance functions for permutations based in these 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). Collectively we will call these distances *interpretation distances*.

In principle, geometric operators for permutations can be defined using these notions of distance. So we can define *rigorously* relative order crossover (ROX) and mutation (ROM), absolute position crossover (APX) and mutation (APM), and adjacency crossover (AX) and mutation (AM). However, ROD, APD and AD are not straightforward to implement *exactly* (producing only matrices corresponding to *feasible permutations*) because are only indirectly related to the syntax of permutations. Therefore these distances are not a natural choice to be the base for geometric crossover. We will see later that to make crossover operators that are straightforward to implement over the syntax at hand we need to use distances that are firmly rooted in the solution representation (read edit distances).

Each interpretation distance is also associated with a notion of *syntactic non-edit* distance between permutations. These distances are based on the syntax of permutations but do not rely on the notion of edit move. ROD is the *displacement distance* that sums for each element the number of positions away is in the two permutations. APD is the *Hamming distance* for permutations. AD is the *breakpoint distance* that counts the occurrences of two elements being consecutive in a permutation *and* non-consecutive in the other. In principle, geometric operators can be defined using these notions of syntactic distance but again, because these distances are not defined *operationally*[3] over the syntax, the definitions of the corresponding genetic operators do not tell us *how to manipulate the syntax* of permutations to implement them producing *feasible permutations* within a segment.

Since edit distances have a fundamental role in the geometric framework we analyse them in details in the next section. However, since interpretation distances (based on matrices), syntactic distances and edit distances are interdependent, all of them turn out to be useful and their connection with edit distances can be exploited in various ways as we show in later sections.

## 4. Edit distances and mutations

Both edit distances and traditional mutation operators are based around the notion of edit move. In the following we report the most common *edit moves* for the permutations (Vergara, 1997). Two requirements for an edit move (to be the base for an edit distance that respects the metric axioms) are *symmetry*, if the configuration *a* is reachable from configuration *b* in one move then *b* must be reachable from *a* in one move too, and *full-connectivity* that states that any configuration has to be reachable from any other in a finite number of moves.

[3] Since edit moves are unitary syntactic transformations that preserve *syntactic feasibility*, crossover operators that are defined using edit distances can be implemented as a sequence of unitary syntactically feasible transformations. This suggests how to reach feasible offspring from feasible parents. Moreover, depending on the syntax and the edit move used, the composition of unitary syntactic transformations can be aggregated in a single-shot syntactic manipulation (a non-unitary edit move) producing the same effect and still preserving feasibility. This suggests how to implement crossover operator quite straighforwardly. This is an important property of edit distances that other syntactic distances do not have.

- *Inversion (or block-reversal):* The reversal move selects any two points along the permutation then reverses the subsequence between these points. This move is particularly well-suited for the TSP and for all the problems that naturally admit a permutation representation in which adjacency among elements plays an important role.
- *Insert and block-transposition:* The insert move selects one element and inserts it at some other position in the permutation shifting elements to make room for the insertion. The block-transposition move inserts a sub-sequence of elements rather than a single element. These moves have been used successfully for scheduling problems in which relative order of elements is important.
- *Swap and adjacent swap:* The swap move selects two elements and swaps their positions. The adjacent swap operator swaps two contiguous elements.
- *Scramble*: This move selects a sub-list and randomly reorders the elements while leaving the other elements in the permutation in the same absolute position.
- *Constrained moves:* there are a number of variations for each of these moves which result from imposing constraints on the edit move such as the maximum size of the block or a limit on the distance between origin and destination points.

All the previous edit moves are symmetric and fully connected. For example, the inversion move is symmetric because re-reversing the same sub-list produces the original permutation. It is connected because by repeated reversions it is possible to reach any permutation from any other permutation. Also the adjacent swap move is symmetric and connected because bubble sort based on adjacent swap is able to sort any permutation of elements. The same holds for the swap operator. The adjacent swap operator can be seen as a special case of swap or as a two-element sub-list inversion move.

The *edit distance* between any two syntactic configurations is the minimum number of edit moves necessary to transform one configuration into the other. This definition of distance respects the axioms for a metric: identity, symmetry and triangular inequality. Each notion of edit move gives rise to a notion of edit distance. Therefore we will talk of *reversal distance*, *transposition distance*, *swap distance*, *adjacent swap distance*, *scramble distance* and so on.

The most common mutation operators for permutations mirror almost exactly the most common edit moves. Even if the very idea of mutation is very similar to the one of edit move, we want to emphasise that an edit move is a deterministic and unitary syntactic transformation whereas a mutation is a non-deterministic operator with a given probability distribution and does not need to correspond all the times to a single edit move. In fact, there are mutations that are local in their effect (point mutation) and mutations that are non-local with a decreasing probability distribution on the distance (for example the standard mutation for binary string under Hamming distance is of this type). Furthermore, there are mutations that have a non-symmetric definition, for example a mutation that allows for insertion of an element in a permutation only after the current position of the element itself.

## 5. Crossovers for permutations and sorting algorithms

The definition of genetic operators we have introduced is geometric and based on a notion of distance. The distance can be any distance that respects the metric axioms.

Hence, given any distance the genetic operators are well-defined and fully-specified. The permutation representation allows for distances of various types and with various origins. We have considered a few in sections 3 and 4. What type of distances should we use to specify genetic operators?

Although in theory any distance will do, in practice the genetic operators need to be implemented and so they require a notion of distance that is defined around a representation and that is strongly rooted in the syntax like edit distances. In section 2, we have emphasized the practical importance/necessity of specifying the distance based on the representation. Indeed, when the distance is an edit distance it becomes possible to specifying operationally the genetic operators. However, specifying the genetic operators using other types of distance can be useful too: interpretation distances can be used to guiding design by suggesting a convenient edit distance to use for the problem at hand (see section 8) and syntactic non-edit distances can be used as approximations or bounds of edit distances and can be helpful in implementations and in proving theoretical results.

Since the notion of edit distance is defined through the syntax of the specific representation is rooted in, geometric objects/transformations defined using such a distance have a natural correspondence to syntactic objects/transformations peculiar of the specific representation and move considered (distance duality). In the specific case of permutations, 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 implementing such crossovers by *sorting algorithms*. Bubble sort and insertion sort fit the definition of geometric crossover for, respectively, the adjacent swap distance and the swap distance. So, *some ordinary sorting algorithms can actually be used as crossovers*! Some edit distances give rise to crossovers that can be implemented exactly and efficiently. Other edit distances give rise to crossovers that are possible to implement efficiently (in polynomial time) only in an approximated way (see TSP example in section 10). For example, constraints on edit moves transform the complexity of computing the edit distance, hence of crossover, from polynomial to NP-hard (Vergara, 1997).

Not all classic sort algorithms fit the definition of geometric crossover though. Only those algorithms that use the same move throughout the sorting and are guaranteed to do always the minimum number of move applications. Technically, these algorithms when applied to permutations solve the "minimal permutation sorting by $x$ problem" where $x$ stands for the move used. Bubble-sort, insertion sort and selection sort belong to this class of sorting algorithms 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 geometric crossovers (but they could still lead to good recombinations for permutations).

In the following we highlight five important differences between sorting algorithms and crossover operators:

(i) Crossover operators sort one parent permutation *toward the order of the other parent*, not to the fully ordered permutation (1 2 3 …). However, it is easy to prove that these two ways of sorting are, in fact, equivalent.

(ii) Crossover operators do not sort a vector of real numbers or a list of words: they *sort permutations* in which the rank of each element to sort is already known. This additional information can be used to build different and more efficient sorting algorithms that do not resemble the classical (and more general purpose) ones.

(iii) Crossover operators *sort permutations around the notion of moves*. Like classical sorting algorithms, they are optimal on the numbers of moves employed. Unlike them, crossover operators do not require optimality in the number of comparisons too.

(iv) Crossover operators *interrupt the sort* at a random point to produce the offspring.

(v) Crossover operators may perform a *non-deterministic sorting*. That is, given the same permutation to sort, crossover may follow different sorting trajectories in different executions[4].

## 6. Genetic operators implementation

From an implementation point of view genetic operators need to be easy and efficient to implement. We propose the following operators.

*Generalization of standard binary string mutation*

In Figure 3 we report a straightforward generic mutation that can be used with any solution representation coming with a notion of edit distance. $p_m$ 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 a decreasing probability depending on the distance. It is therefore a macro-mutation operator and it is still a geometric ε-mutation operator (its image set is in a ball with radius the diameter of the space).

**MUTATION**

**Input**: *parent*, $p_m$ **Output**: *offspring*
*offspring* ⟵ *parent*
**WHILE** $p_m >$ *uniform_random*([0,1]) **DO**
    *neighbours* ⟵ *get_neighbours*(*offspring*)
    *offspring* ⟵ *uniform_random*(*neighbours*)
**END**
**RETURN** offspring

Figure 3 – Mutation operator

[4] There are two quite 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 (identity permutation). 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.

*Sorting crossovers*

In figure 4 we report the pseudo-code for all sorting crossovers for permutation. First the two parent permutations are composed obtaining a permutation in normal form. This has the advantage of transforming the less ordinary task of sorting one permutation toward a second arbitrary permutation into the more standard task of sorting a permutation into a completely order permutation (identity permutation).

Then any sort algorithm that satisfies the requirement of being a "minimal sorting by *x*" algorithm, can be used to sort the permutation in normal form taking care of collecting the sequence of sorting moves (sorting trajectory). In Section 9, we have tested three classical sorting algorithms in their deterministic and uniform non-deterministic versions to the n-queens problem.

The distance between the two parents permutations based of the move considered is the number of moves on any (minimal) sorting trajectory between the two permutations. 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.

Geometrically, given the two parents, the offspring of a deterministic sorting crossover all lay on the same geodesic connecting them; those of a non-deterministic sorting crossover instead are spread across all segment (the union of all geodesics connecting two points) identified by the two parents. In this respect, the former resembles more the traditional one-point crossover for binary strings; the latter instead resembles the uniform crossover. However, the actual probability distributions over the offspring are not necessarily uniform and depend on the specific geometry of the space considered.

**SORTING CROSSOVER**

**Input**: *parent_perm1*, *parent_perm2* **Output**: *offspring_perm*

*normal_perm* ⟵ *parent_perm1* ° *parent_perm2* $^{-1}$
*sort_moves_seq* ⟵**sort** (*normal_perm*)
*distance*⟵ *length* (*sort_moves_seq*)
*crossover_point* ⟵ *uniform_random*(*integer range*: [0, *distance*])

*offspring_perm* ⟵ *parent_perm1*
**WHILE** *crossover_point>0* **DO**
    *offspring_perm*⟵ *offspring_perm* ° *sort_moves_seq*(*crossover_point*)
    *crossover_point* ⟵ *crossover_point -1*
**END**
**RETURN** *offspring_perm*

Figure 4 – Sorting crossover

## 7. Do pre-existing crossovers fit the abstract geometric definition of crossover?

### *7.1 Pre-existing crossovers and permutation interpretations*

There are a number of crossover operators defined for permutation (for a good overview, see (Back et al, 2000)). 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* emphasizes the fact that a permutation is seen as a relative order, *cycle crossover* preserves absolute positions, and *edge recombination crossover* 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. For example cycle crossover transmits perfectly the common positional information of parents to children; both Davis's order crossover and edge recombination are not able to transmit perfectly, respectively, the common relative order of the parents and the adjacency relation. However, another crossover, the *merge crossover*, 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. This is indeed possible since the three relations have subtle interdependencies. One of such crossover operators is the *partially matched crossover*. Hybrid crossovers such as these have the advantage to work reasonably well independently from the specific interpretation of the permutation. However, they perform worse than perfect crossovers for a specific interpretation of the permutation on a problem in which this interpretation is relevant.

### *7.2 Fitting absolute position crossovers*

In section 5 we started with the general definition of geometric crossover and, while specifying it to permutation representation, we discovered that geometric crossovers for permutations are sorting algorithms. Now, we do the opposite, we start from pre-existing crossovers and show how these fit our definition of abstract crossover.

Since it is easy to classify crossovers according to the specific interpretation given to a permutation (see Section 7.1), instead of showing that all pre-existing crossovers fit our geometric definition of crossover, it is more sensible to show that all perfect crossovers under a certain interpretation are sorting algorithms according to a specific notion of edit move for permutation.

In this section we give two results about absolute position crossovers. We show that selection sort based crossovers are position preserving crossovers and that a specific pre-existing crossover that is position preserving is a selection sort based crossover. In sections 10 and 11, we study the cases of adjacency crossovers and relative order crossovers in connection with two important and exemplary problems, TSP and JSSP respectively.

*Theorem (Selection sort crossover is position preserving)*
Selection sort crossover recombines two permutations producing offspring permutations preserving common values at the same position.
*Proof*
Selection sort uses the swap move to order the initial permutation to the target permutation. A sorting move for selection sort is one that fixes at least one value of the current permutation to the target position. When the current permutation is different from the target permutation, there always exists at least one sorting move. A swap that does not fix at least one value to its target position is not a sorting move and is not on a minimal sorting trajectory. A swap move that changes a value of the initial permutation that is already in its target position is not a sorting move because, instead of fixing at least one value, it un-fixes at least one. Hence all the intermediate permutations on a sorting trajectory (offspring) share the common values at the same positions with the initial and target permutations (parents). ■

*Cycle crossover is a selection sorting crossover*
Cycle crossover divides the elements of the two parent permutations into cycles and then creates offspring by selecting randomly successive cycles from each parent. 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. Cycle crossover preserves perfectly the common absolute position of elements from parents to offspring.

*Theorem*
The cycle crossover can be seen as a selection sort crossover.
*Proof*
The procedure to exchange one cycle at random between the two parent permutations is as follows. Pick a position in the initial permutation at random. The value at that position is the reference value. Swap the reference value with the value at the same position in the target permutation. This is a sorting move because at least one value at one position is fixed by this move. If the reference value is not fixed swap the reference value with the value at the same position in the target permutation and keep on iterating this move until the reference value is fixed as well. All these moves are on a minimal sorting trajectory since each one fixes at least one value. When the reference value is also fixed the current permutation differs from the initial permutation exactly of one cycle, which has been substituted by the corresponding cycle of the target permutation by the sequence of swaps. This differs from the original cycle crossover implementation only in the fact that, instead of first identifying a cycle and then exchange it all in one go, it does swap moves until a whole cycle is exchanged without making it explicit beforehand. ■

Pre-existing crossovers that fit the definition of geometric crossover are sorting algorithms in disguise! Indeed they produce offspring on the minimal sorting trajectory between parents according to some edit move. The fact that most of pre-existing crossover at first glance do not look like sorting algorithms, even when they are pure crossovers for a given interpretation of the permutation, is because they are special implementations of *sorting algorithms for permutations* and do not apply to general vectors of number. Since every element of a permutation identifies also its rank in the completely ordered permutation, this information can be used to

implement sort algorithms for permutations that are more efficient than standard sort algorithms and do not quite look the same.

The case of cycle crossover being a perfect crossover under a certain interpretation of permutation and happening to be also a geometric crossover (read sorting crossover) is not accidental. Permutation interpretations and geometric aspects are intimately connected. In Table 1, we propose a summary of important attributes of crossover/recombination operators for permutations classified by permutation interpretation.

Table 1 – Attributes of crossovers for permutations (summary)

| Permutation Interpretation | Edit Move | Sorting Algorithm | Computational Complexity | Problem Example | Crossover Example | Syntactic Distance |
|---|---|---|---|---|---|---|
| Position | swap | Selection Sort | P | n-queens | Cycle crossover | Hamming |
| Adjacency | block-reversal | Sorting by Reversals | NP-hard | TSP | Edge recombination | Breakpoints |
| Relative order | adjacent swap | Bubble Sort | P | JSSP | Merge crossover | Displacement |
| Hybrid | - | - | P | general | PMX | - |

## 8. Design issue: matching problem and algorithm

One peculiarity of the permutation representation is that there are many equivalently natural notions of crossover; hence one may wonder what the right notion of crossover is after all. For the NFL theorem (Wolpert & Macready, 1996) there cannot be the right crossover: what is a good crossover for a problem must be necessarily bad for another one. The question is therefore how to match crossover and problem in a principled way. We discuss three heuristic methods to do that: build a crossover using a good mutation, build a crossover using a neighbourhood based on the small-move small-fitness change design principle, or build a crossover using a distance that is connected with a distance that is relevant for the solution interpretation.

### *8.1 Good mutation, good crossover*

A certain general way of matching problem and evolutionary algorithm using crossover is a good one if it produces systematically better performance than random search *in general*. Notice that to match problem and algorithm one has to use *problem knowledge* (and algorithm knowledge) and therefore the general superior performance one gets does not conflict with the NLF theorem that holds true only in a black-box scenario. Therefore the question is: *how can we choose solution representation and edit move to obtain a good match abstract crossover/problem at hand?*

Our hypothesis of a general way of matching an evolutionary algorithm using crossover and problem at hand is heuristic and passes through mutation: if a neighbourhood structure obtained by choosing solution representation and edit move for the problem at hand is *good for mutation* (or simple local search) then it is *also good for crossover*. This hypothesis makes sense since it mirrors the well-known rule of thumb that a good neighbourhood structure for a problem works well with different meta-heuristics (Glover, 2002).

Hence, what makes sense to test experimentally here is our hypothesis: whether or not *in general* a good match mutation/problem-at-hand corresponds to a good match crossover/problem-at-hand. Here we are not testing if *abstract crossover is good in general*, which for the NFL is futile. We are testing a *general conditional statement* that is not affected by the NFL and that is likely to be true since crossover and mutation are related operators.

### *8.2 Permutation interpretations and edit distance design*

Permutations indeed allows for various notions of edit moves. In order to choose the "right" edit move and consequently the "right" crossover, the designer needs to know what the permutation is meant to represent (permutation interpretation) and this piece of information can be obtained only by knowing what the problem is. Therefore the notion of permutation interpretation is inherently connected with the notion of problem knowledge. As we have seen in section 3, permutation interpretations connect subtly with different types of distances for permutations. Exploiting the connection between interpretation distances, or phenotypic distances, that naturally connect with the interpretation of a specific representation for the problem at hand so embed problem knowledge, and edit distances that naturally connect with the syntax of the specific representation so allowing for the design and implementation of "concrete" operators matching "abstract" interpretation based operators, seems to be a promising direction indeed to match problem at hand to specific operators.

The technical instrument necessary to map two distances is the embedding. An embedding maps the points of a given metric space into the points of a host metric space. The embedding has low distortion if all of the interpoint distances are approximately preserved. In applications, the host metric space is "simpler" than the original metric space. For many optimisation problems, low distortion of the interpoint distances causes only a correspondingly low distortion in the value of the objective function: near-optimal solutions with respect to the distorted distances correspond to near-optimal solutions with respect to the original distances; an approximation algorithm designed for the host metric space provides an approximation guarantee for the original metric space.

### *8.3 Minimal change principle*

Depending on the interpretation of the permutation, the same mutation operator 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 (Radcliffe, 1992; Radcliffe, 1994). According to this principle, there are three mutation operators that do a different minimal change in a permutation, one for each interpretation. When the permutation is thought 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. Finally, 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.

A general heuristic way to design good neighbourhood structures for local search and other meta-heuristics and, simply states that neighbours solutions should have similar fitness. This rule of thumb has been rediscovered by many authors in many fields of search and optimisation many times and has various names (Pardalos & Resende, 2002).

## 9. Sorting crossovers for the n-queens problem

The eight queens' puzzle is the problem of placing eight chess queens on an 8 by 8 chessboard such that none of them is able to capture any other. The piece colour is ignored, and any piece is assumed to be able to attack any other. That is to say, no two queens should share the same row, column, or diagonal. The generalized problem of placing *n* "non-dominating" queens on an *n* by *n* chessboard is a good example of a simple but non-trivial constraint satisfaction problem and, for this reason, is often used as an illustrative problem for non-traditional approaches, such as constraint programming, logic programming or genetic algorithms (Russell & Norvig, 2003).

### *9.1 Solution representation, fitness function and genetic operators*

Various encodings and solution representations for the n-queens problem have been suggested (Eiben, 1995); some of them reduce the search space enormously compared to more naïve approaches thereby speeding up the search. Here, we use a permutation to represent a potential solution: the position of an element in the permutation identifies a row in the chessboard and the value of the element itself specifies the column of the location the queen in that row. This sets exactly *n* queens on an *n* by *n* chessboard and reduces the search space, in that, column and row conflicts are eliminated. Only diagonal conflicts may arise. The fitness function is the number of conflicts among pairs of queens and has to be minimized.

The mutations and crossovers we consider are based on the generic mutation operator in figure 3 and on the generic sorting crossover operator in figure 4.

We want to compare crossover operators derived from three classical sorting algorithms: selection sort, bubble sort and insertion sort. As already mentioned in Section 6, these algorithms fit the definition of algorithm doing "minimal permutation sorting by *x*" paradigm; they are based on swap move, adjacent swap move and insertion move respectively. The first two moves are good moves for absolute position and relative order problems, respectively. The third one does not associate to a clear-cut interpretation of the permutation and mixes characters of the previous two.

For each sorting algorithm we propose two types of crossover. The first type *deterministically* sorts the first parent permutation toward the second parent permutation, collects all the permutations on the sorting trajectory and returns one of them at random. The second crossover type chooses a minimal sorting trajectory between two permutations *at random* (non-deterministic sorting) and then, analogously to the previous one, returns as offspring one of the permutations on that trajectory at random. We also consider three types of mutations; one for each move operator.

For reference, we compare the sorting crossover operators with the traditional partially matched crossover (PMX) recombination with swap mutation that has been shown to perform really well for this problem (Goldberg, 1989).

### *9.2 Experiments*

In our experiments we used an evolutionary algorithm with the parameters shown in Table 2. The results of the experiments are shown in Figures 5 and 6.

Table 2 – Parameters of the evolutionary algorithm

| | |
|---|---|
| Problem size | *100* |
| Population size | *5000* |
| Mutation probability | *0.1* |
| Crossover probability | *0.5* |
| Generation | *500* |
| Selection | *tournament size 2* |
| Statistics | *Average 30 runs* |

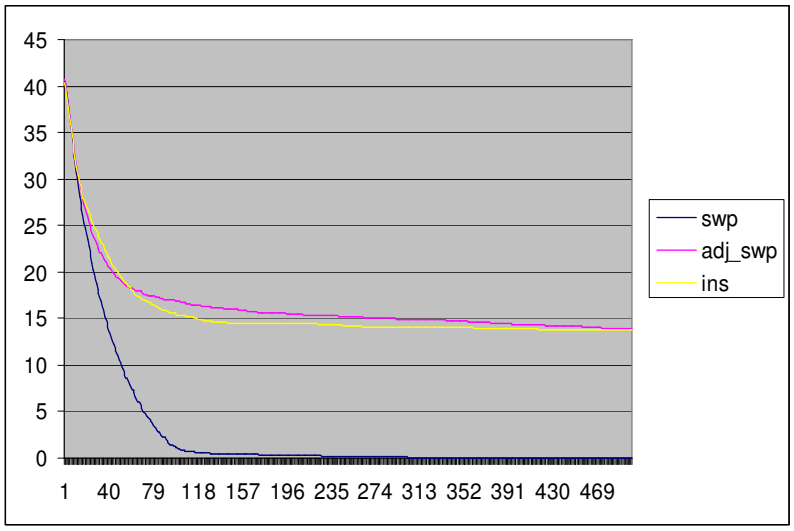


*Figure 5 – Comparison among mutations*

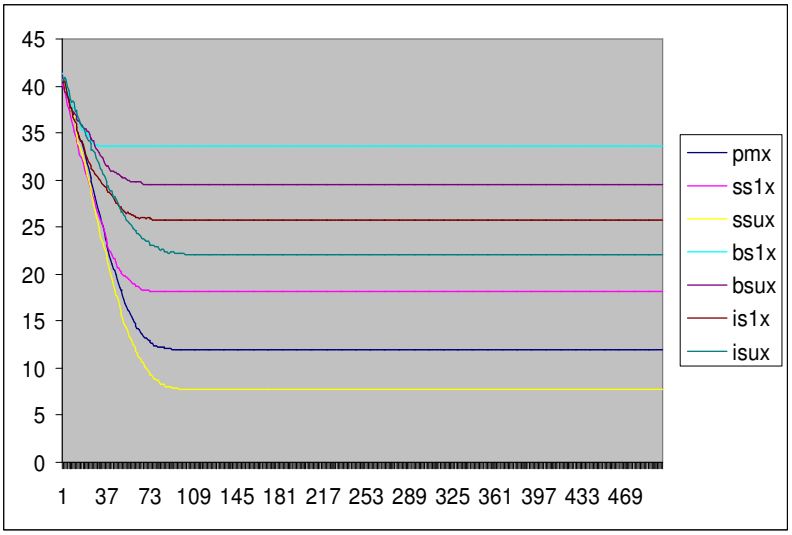


*Figure 6 – Comparison among crossovers*

Figure 5 compares the algorithm using three different types of mutation, swap mutation (swp_only), adjacent swap mutation (adjswp_only) and insertion mutation (ins_only) without crossover. The x axis is number of generations and the y axis is the fitness of the best individual in the population averaged over 30 runs. Since the objective function has to be minimized, the swap mutation is clearly much superior of the other two mutations that perform similarly, perhaps with the insertion mutation performing slightly better.

Figure 6 compares the performance of the algorithm using the following crossovers: partially matched crossover (PMX), deterministic and non-deterministic selection sort crossover based on the swap move (SS1X, SSUX), deterministic and non-deterministic bubble sort crossover based on the adjacent swap move (BS1X, BSUX), deterministic and non-deterministic insertion sort based on the insertion move (IS1X, ISUX). No mutation was used. The picture gives a clear ranking of the performance of crossovers: (1) SSUX, (2) PMX, (3) SS1X, (4) ISUX, (5) IS1X, (6) BSUX and (7) BS1X.

Each non-deterministic sorting crossover outperforms the corresponding deterministic one. This is not necessarily due to an inherent superiority of non-deterministic sorting crossovers over the deterministic counterparts. Non-deterministic crossovers are probably advantaged since they are more explorative and compensate the lack of mutation.

The hypothesis that a move that produces a good mutation operator produces also a good corresponding crossover operator is clearly corroborated by the experiment. Indeed in our rank of crossovers both SSUX and SS1X, based on the swap move, are superior to both ISUX and IS1X, based on the insertion move, which in turn are superior to both BSUX and BS1X, based on the adjacent swap move. This classification mirrors exactly the classification of mutations on performance.

We like to highlight that the rule “good mutation implies good crossover” is presumably a good heuristics that, nevertheless, neglects the geometric details of the space on which the operators are defined; so it may fail. Further study, theoretical and

experimental, is needed to understand the impact of specific geometric properties of the space to the relation between the performance of mutation and crossover.

Interestingly, the non-deterministic sorting crossover based on the swap move (SSUX) outperforms the PMX operator that is one of the best operators for the n-queen problem. This shows the power and the simplicity of the geometric framework to build new effective crossovers for the problem at hand.

## 10. Adjacency crossover, circular permutations and TSP

Edge recombination is an operator expressly designed for TSP. It considers a solution as a tour of cities and, therefore, rather than being defined for permutations 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 (Back, 2000).

As in the linear permutation case, also for circular permutations it is possible to write 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 if we could implement it, it would actually achieve what *edge recombination can only aspire to.*

A distance based on adjacency matrix is not an edit distance. In the case of circular permutations, the block-reversal move is the notion of edit distance closest to the adjacency matrix distance because in a single application to a tour, this does the minimal change to the adjacency relation among elements in the permutation. This move is the well-known *2-opt* move, and it is the basis for successful local search algorithms for TSP (Glover, 2002). Figure 7 shows the possible offspring (the segment) between two circular (parent) permutations under topological crossover.

Analogously to the linear case, the circular permutations in the segment under block-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 (Solomon et al, 2003). So, *the topological crossover under this notion of distance cannot be implemented efficiently*.

(Solomon et al, 2003) showed that 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 also for the former. Sorting linear permutations by reversals is NP-hard too (Caprara, 1997). However a number of approximation algorithms exist to solve this problem within a bounded error from the optimum (Kececioglu & Sankoff, 1995)[5]. So, although an efficient implementation of geometric crossover cannot exist, it is possible to

[5] Sorting linear and circular permutations by reversals within a bounded error from the optima is a non-trivial task and has been object of research in the last decade since has important applications in genetics. Papers on the topic are extremely technical and rather than giving the full algorithm to solve the problem, they prove some properties of distances that can be used to implement the algorithm.

implement approximate crossovers whose image set is a super-set of that of the exact crossover. Further research will need to investigate these and compare them to ERX.

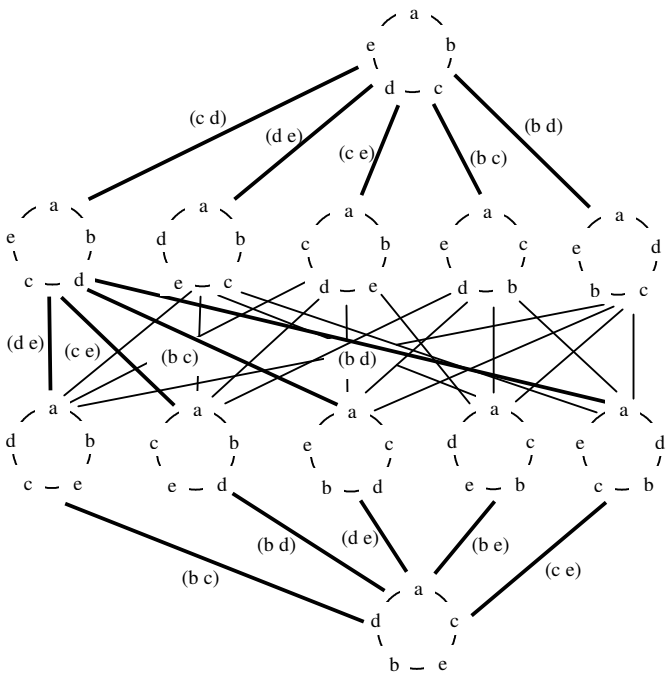


*Figure 7- Example of geometric crossover between two circular permutations*

### *10.1 Discussion: computational barrier*

There is a *computational barrier* that manifests itself when the edit move is not elementary enough. The crossover operator cannot be implemented efficiently and exactly.

However the situation is more complicated. You can think also of complicating artificially the syntax of the representation and the corresponding move making it appearing not as a unitary move but as a major move, but in actuality the corresponding neighbourhood structure is the same as before (so the two pair representation and move are *equivalent* from a point of view of the search) and therefore the crossover has the same computational complexity. Indeed the complexity of crossover is connected with the *simplest* pair representation and move.
The computational barrier seems to be a limitation of the topological crossover: what is the use of a nice and general operator that cannot be implemented efficiently? There are two considerations to do: (i) if you choose a simple enough move definition the crossover can be implemented efficiently (ii) but more importantly, the topological crossover being defined over the structure of the search space shows an *interesting connection between geometry of the search space and computational complexity of the problem at hand*. This has in turns two consequences:

From a computational complexity point of view and therefore from the efficiency point of view of the evolutionary algorithm employing a specific operator, the complexity of the operator can be a major issue. This is not true for actual algorithm,

that the practitioner uses and makes sure that the operator is very efficient, but this is a warning for the theoretician that defines operators *assuming* that they are implementable efficiently and focuses the analysis of efficiency of the algorithm based on other aspects.
If for a given hard problem you need a specific neighbourhood structure to solve it efficiently, it is not surprising that you are *moving* the inherent complexity of the problem from the search process to the actual implementation of the search operator. In other words, you could define an operator that in a polynomial number of steps solves a NP-Hard problem. That is unlikely to mean that NP=P but rather that your operator is going to be NP-Hard to implement! This could be the case of topological crossover. So the moral is that you need a trade-off between the efficiency of the move you choose (so that the crossover can be implemented in polynomial time) and its effectiveness (so that you will not solve a NP-hard problem to optimality in a polynomial number of applications of the crossover derived by the chosen move). This is not due to a limitation of the topological crossover, but rather to the inherent computational complexity of the problem at hand.

## 11. Relative order crossover, partial order relationship and JSSP

Job Shop Scheduling Problem is a strongly NP-hard combinatorial optimisation problem and one of the best known machine scheduling problems. The JSSP is characterised by being highly constrained and ordering problem. Both these aspects have to be considered in order to figure out a natural solution representation to employ with an evolutionary algorithm. The main difficulty in choosing a proper representation for highly constrained combinatorial optimisation problems is dealing with the infeasibility of the solutions produced during the evolutionary process. In (Moraglio, 2000) was proposed an evolutionary algorithm that uses a natural encoding of solutions using a simple permutation representation, which covers all and only the feasible solution space, and a class of order-preserving recombination operators that guarantee the transmission of meaningful characteristics together with preserving the feasibility of solutions. Computational experiments showed that this representation outperforms many others producing good quality solutions in less time. Here we are interested in showing that this *class of recombination operators fit the definition of abstract crossover* under adjacent swap distance (hence one could use bubble-sort crossover for JSSP) and that *the natural encoding JSSP solution space to permutation space has a simple geometric interpretation*.

### *11.1 Disjunctive graph: problem constraints structure representation*

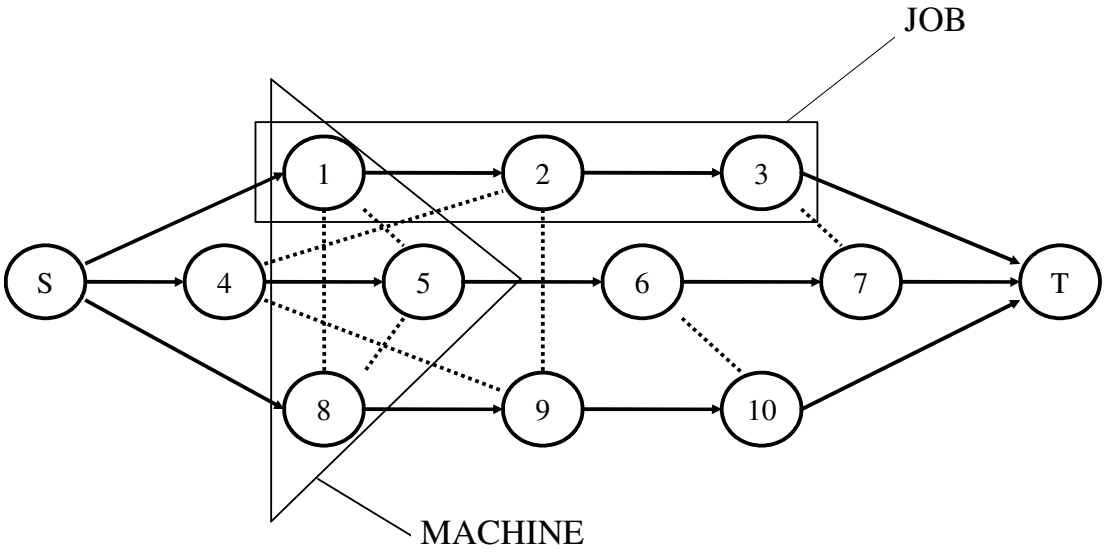


*Figure 8 – Disjunctive Graph (Elements of A are indicated by arrows and elements of E are indicated by dashed lines.)*

The job shop scheduling problem can be represented with a disjunctive graph (Brucker, 2001). A disjunctive graph $G=(N, A, E)$ is defined as follows: $N$ is the set of nodes representing all operations, $A$ is the set of arcs connecting consecutive operations of the same job, and $E$ is the set of disjunctive arcs connecting operations to be processed by the same machine. A disjunctive arc can be settled by either of its two possible orientations. The construction of a schedule will settle the orientations of all disjunctive arcs so as to determine the sequence of operations on the same machine. Once a sequence is determinate for a machine, the disjunctive arcs connecting operations to be processed by the machine will be replaced by the oriented precedence arrow, or conjunctive arc. The set of disjunctive arcs $E$ can be decomposed into cliques, one for each machine. The processing time for each operation can be seen as a weight attached to the corresponding nodes. JSSP is equivalent to finding the order of the operations on each machine, that is, to settling the orientation of the disjunctive arcs so that the resulting solution graph is acyclic (there are no precedence conflicts between operations) and the length of the longest weighted path between the starting and terminal nodes is minimal. This length determines the *makespan*.

### *11.2 Permutation representation and natural solution encoding*

Many solution encoding/representation for JSSP have been proposed. The presence of constraints creates infeasibility problems for many of them. Depending on the representation chosen, there are various problems that may arise: creating solutions with cycle of constraints, creating solutions that do not respect the problem constraints, or using encoding that builds always feasible solutions but that does not cover all the solution space leaving out the optimum. Further important (soft) constraints of a encoding/representation is allowing for genetic operators that combine and transmit meaningful sub-parts of solutions.

The problem constraints structure (solid arrows in Fig. 8) can be seen as a partial order among operations (nodes in Fig. 8). A schedule, a feasible solution for the specific problem, is obtained by orientating all the dashed lines in a way to obtain an acyclic graph; this as well can be interpreted as a partial order among operations that is stronger and include the partial order induced by the problem constraints alone. While adding further arrows to the solution graph, taking care not to create cycles, we obtain a total order among operations. This does not need to be specified by a graph; a simple sequence (a permutation) of nodes is appropriate. For example, by first specifying a solution and then by further specifying order constraints up to saturation we may obtain the permutation *(4, 8, 5, 1, 2, 9, 6, 3, 7, 10)*. This sequence contains all the information necessary to build the constraint structures of the corresponding solution (and also the original problem) and therefore can be used as solution representation. Notice that all solutions can be represented (and in general each solutions has more than one representation), cyclic solutions cannot be represented but solution not respecting problem constraints can be represented.

### *11.3 Common relative order preserving operators*

Recombining permutations corresponding to feasible solutions (respecting the problem constraints) using *operators that perfectly transmit the common relative order*, we obtain the twofold advantage of (i) producing feasible solutions, because the partial order relation induced by the problem is common to both parents and therefore transmitted completely (proven formally), and (ii) combining partial order relationships that are a natural way of decomposing the constraint structures of the solutions. In (Moraglio, 2000) was proposed various operators matching this requirement.

In the following we prove that you can use bubble sort with JSSP.

*Theorem (Bubble sort crossover for JSSP)*
*Crossover*: Bubble sort crossover recombines two feasible JSSP schedules encoded in permutations and produces feasible offspring permutations.
*Mutation*: any adjacent swap over a feasible permutation of two operations not sharing the same job produces a feasible offspring permutation.
*Proof*
Let us consider any two adjacent operations, say *a* and *b*, in a permutation representing a schedule such as *a* precedes *b*. This precedence is due for one of the following mutually exclusive reasons: (i) *a* precedes *b* and *a* adjacent *b* in a job (problem constraint) (ii) *a* precedes *b* and *a* adjacent *b* on a machine (solution constraint) (iii) *a* precedes *b* and *a* adjacent *b* because of the linearity of the permutation (dummy constraint). If you swap *a* and *b* the permutation will correspond to an infeasible solution only when their precedence constraint is a problem constraint (case (i)). In the other two cases, the swap is safe because, locally the swap do not clash with any problem constraint, plus, the swap of two adjacent operations has only a local effect and does not affect the precedence among any other pair of operations; hence from a feasible permutation another feasible permutation is obtained.

Any intermediate permutation on a minimal sorting trajectory of the bubble sort is obtained by adjacent swaps. In particular any adjacent swap performed by bubble sort is a sorting swap; this means that adjacent operations are swapped only when in the target permutation the same two operations have opposite order (do not need to be

adjacent though). For hypothesis we recombine feasible permutations, hence in both permutations initial and target, the precedence constraints on job are always concordant since the two permutations encode solutions under the same problem constraint structure. Therefore none of the sorting swaps of the bubble sort will ever change the relative order of the job constraints hence producing on its sorting trajectory only permutations respecting the job precedence structure. Any of such permutations is a feasible offspring.■

A natural neighbourhood structure over feasible schedules can be defined easily: two feasible schedules are neighbours if one can be obtained from the other by reversing the order of a single couple of operations on the same machine (reversal that do not force to reverse any other couple of operations to maintain feasibility and therefore these operations are adjacent). The distance between two feasible schedules can be induced by such a neighbourhood structure (shortest path distance) hence we have a metric space defined over feasible schedules. This metric space is called adjacent swap metric space over feasible schedules. The same metric space is well-defined over any set of order relationship (read direct acyclic graphs) sharing the same set of edges and differing only in their orientations (arcs).

*Theorem (permutations-schedules betweenness preserving)*
The solution encoding, mapping permutation space and schedule space under adjacent swap metrics, is a betweennes preserving morphism.
*Proof*
Let be $a$, $b$ and $c$ three permutations, such as $c$ in $[a,b]$ under adjacent swap distance, representing respectively the feasible schedules $a'$, $b'$ and $c'$. Let us consider the permutations $a$, $b$ and $c$ as graphs in which the order relation between any two nodes is specified by the presence of an oriented arc. By removing the arc between any same two nodes in the three graphs, obtaining the new graphs $a"$, $b"$ and $c"$, we still have that $c"$ in $[a",b"]$ under the adjacent swap metric defined for these graphs. This is true because (i) either the orientation of the arc was concordant in the three graphs and consequently all the distances between $a"$, $b"$ and $c"$ are the same as the distances between $a$, $b$ and $c$ so that $c$ in $[a,b]$ implies $c"$ in $[a",b"]$; (ii) or the orientation of the arc was discordant in $a"$ and $b"$, implying that in $c"$ it was concordant with either $a"$ or $b"$, and consequently the removal of the arc reduced by one both the distance between $a$ and $b$ and the sum of the distances from the $a$ and $b$ to $c$ so that $c$ in $[a,b]$ implies $c"$ in $[a",b"]$. Since $c$ in $[a,b]$, the case in which the orientation of the arc was concordant in $a$ and $b$ and discordant in $c$ is not compatible with the fact that $c$ is on the minimal sorting trajectory between $a$ and $b$. This reasoning it applies to the removal of arcs up to transforming the permutations $a$, $b$ and $c$ into the corresponding schedules $a'$, $b'$ and $c'$ and consequently $c$ in $[a,b]$ implies $c'$ in $[a',b']$. ■

A natural and effective crossover for JSSP fits perfectly the definition of abstract crossover. This is another piece of evidence in favour of the hypothesis that the abstract geometric definition of crossover captures the notion of crossover in general. In this case we did not use a crossover defined over the solution representation space directly (schedule) but we encoded schedules in permutations. However we have shown that, since the encoding is *betweenness preserving*, the crossover on permutations corresponds to a crossover over schedules in their original domain.

## 12. Conclusions

In this article we have made a long journey started with general theoretical considerations about the abstract framework, continued by applying the theory to the permutation representation, and finishing by considering important well-known problems connected with the permutation representation and further elaborating the consequences of the framework in their specific contexts. We believe we have been able to cover all the major aspects of the permutation representation domain and address them nicely within our abstract geometric framework.

## References


Back et al, 2000. T. Back, D. B. Fogel, Z. Michalewicz (eds). Evolutionary Computation 1. IoP, 2000.

Back et al, 1997. Fogel, T. Back, D. B. Fogel, Z. Michalewicz (eds). Handbook of Evolutionary Computation. Oxford press, 1997.

Brucker, 2001. Peter Brucker. Scheduling Algorithms, Springer.

Caprara, 1997. Caprara, A., Sorting by reversals is difficult, 75--83. In: Proc. of the 1st Ann. International Conference on Computational Molecular Biology (RECOMB 97), ACM, New York, 1997.

Culberson, 1995. J.C.Culberson. Mutation-crossover isomorphism and the construction of discriminating functions. Evol. Comp., 2:279-311, 1995.

Descartes, 1637. R. Descartes. Discourse on Method. 1637.

Davis, 1991. L. Davis, Handbook of Genetic Algorithms, van Nostrand Reinhold, New York, 1991.

Deza & Huang, 1998. M. Deza and T. Huang, Metrics on permutations, a survey, J. Combinatorics, Information and System Sciences 23 (1998).

Eiben, 1995. A.E. Eiben. Solving constraint satisfaction problems using genetic algorithms. WCCI1 proceedings, pages 542-547, 1994

Fox & McMahon, 1991. Fox B R and McMahon M B, Genetic Operators for Sequencing Problems In G J E Rawlins, editor, Foundations of Genetic Algorithms, pages 284-300. Morgan Kaufmann, 1991.

Gitchoff & Wagner, 1996. P. Gitchoff, G. P. Wagner. Recombination Induced HyperGraphs: A New Approach to Mutation- Recombination Isomorphism. Journal of Complexity, 37-43, 1996.

Glover, 2002. F. W. Glover (ed). Handbook of Metaheuristics. Kluwer, 2002.

Goldberg, 1989. D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, MA, 1989.

Jones, 1995. T. Jones. Evolutionary Algorithms, Fitness Landscapes and Search. PhD dissertation, University of New Mexico, 1995.

Kececioglu & Sankoff , 1995. J. Kececioglu and D. Sankoff. Exact and approximate algorithms for sorting by reversals, with applications to genome rearrangement. Algorithmica, 13:180--210, 1995.

Langdon & Poli, 2002. W. B. Langdon, R. Poli. Foundations of Genetic Programming. Springer, 2002.

Moraglio, A. 2000. Genetic Local Search for the Job-Shop Scheduling Problem. Master thesis, Politecnico di Torino, Torino, Italy.

Moraglio & Poli, 2004. Topological interpretation of crossover, Proc. of GECCO 2004.

Pardalos & Resende, 2002. P. M. Pardalos, M. G. C. Resende (eds) Handbook of Applied Optimization. Oxford University Press, 2002

Radcliffe, 1992. N. J. Radcliffe. Nonlinear genetic representations. In R. Manner and B. Manderick, editors, Proceedings of the 2 nd Conference on Parallel Problems Solving from Nature, pages 259-268. Morgan Kaufmann, 1992.

Radcliffe, 1994. N. J. Radcliffe, 1994. The Algebra of Genetic Algorithms, Annals of Maths and Artificial Intelligence 10, 339 --384.

Radcliffe, N. J., Surry, P. D. Fundamental limitations on search algorithms: evolutionary computing in perspective." Computer Science Today: Recent Trends and Developments (ed. J. van Leeuwen), 275-291. Lecture Notes in Computer Science 1000. Springer-Verlag, Berlin, 1995.

Reidys & Stadler, 2002. C. M. Reidys, P. F. Stadler. Combinatorial Landscapes. SIAM Review 44, 3-54, 2002.

Russell & Norvig, 2003. S. Russell & P. Norvig. Artificial Intelligence - A Modern Approach, Prentice Hall.

Stephens & Poli, 2004 R. Stephens and R. Poli. EC Theory “in Theory”: Towards a Unification of Evolutionary Computation Theory. In A. Menon (ed), Frontiers of Evolutionary Computation, pp. 129-156, Kluwer, Boston, 2004.

Solomon et al, 2003. Andrew Solomon, Paul Sutcliffe, Raymond Lister: Sorting Circular Permutations by Reversal. . WADS 2003: 319-328

P.D. Surry and N.J. Radcliffe. Formal algorithms + formal representations = search strategies. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, Proceedings of the 4 th Conference on Parallel Problems Solving from Nature, number 1141 in LNCS, pages 366-375. Springer Verlag, 1996.

Surry, 1998. P. D. Surry. A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms. PhD dissertation, University of Edinburgh, 1998.

Syswerda, 1989. G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Scha er, editor, Proceedings of the International Conference on Genetic Algorithms, San Mateo (CA), 1989. Morgan Kaufmann Publishers.

Vergara, 1997. J P C Vergara, Sorting by Bounded Permutations, Virginia Polytechnic Institute, PhD thesis, 1997.
Van der Vel, 1993. M. van de Vel. Theory of Convex Structures, Elsevier, Amsterdam, 1993.
Whitley, 1994. D. Whitley. A Genetic Algorithm Tutorial. Statistics and Computing (4):65-85, 1994.
Wolpert & Macready, 1996. D. H. Wolpert, W. G. Macready. No Free Lunch Theorems for Optimization. IEEE Transaction on Evolutionary Computation, April 1996.