Geometric Crossover for Permutations with Repetitions: Application to Graph Partitioning

Alberto Moraglio[1], Yong-Hyuk Kim[2], Yourim Yoon[3],
Byung-Ro Moon[3], and Riccardo Poli[1]

[1] Department of Computer Science, University of Essex
Wivenhoe Park, Colchester, CO4 3SQ, UK
{amoragn, rpoli}@essex.ac.uk
[2] Department of Mathematical Sciences, Seoul National University
Sillim-dong, Gwanak-gu, Seoul, 151-747 Korea
yhdfly@soar.snu.ac.kr
[3] School of Computer Science & Engineering, Seoul National University
Sillim-dong, Gwanak-gu, Seoul, 151-744 Korea
{yryoon, moon}@soar.snu.ac.kr

Abstract. Geometric crossover is a representation-independent generalization of the traditional crossover defined using the distance of the solution space. By choosing a distance firmly rooted in the syntax of the solution representation as basis for geometric crossover, one can design new crossovers for any representation. In previous work we have applied geometric crossover to simple permutations. In this paper we design a new geometric crossover for permutations with repetitions that naturally suits partition problems and test it on the graph partitioning problem. Our new crossover outperforms all previous ones.
Keywords: Geometric crossover, cycle crossover, permutations with repetitions, graph partitioning.

1 Introduction

Geometric crossover [20] is a representation-independent operator defined over the distance of the search space. Informally, geometric crossover requires the offspring to lie between parents.

The formal definition of geometric crossover enables us to build a representation-independent theory of evolutionary algorithms: it can be shown that an evolutionary algorithm endowed with geometric crossover and selection does a form of convex search regardless of the underlying solution representation, fitness landscape, specific distance, probability distribution of the offspring and selection mechanism. Knowing how geometric crossover searches the search space is very important because this allows us to identify a general class of fitness landscapes for which it works well.

Despite its simple geometric definition, geometric crossover captures the notion of "real-world" crossover: geometric crossover generalizes many pre-existing

search operators for the major representations used in evolutionary algorithms, such as binary strings and real vectors [20], permutations [22], syntactic trees [21] and sequences [23].

The formal definition of geometric crossover can also be used to guide the design of new specific crossover operators for non-standard representations using as base for geometric crossover distances rooted on the specific representation (e.g., edit distances) [22].

Traditional crossovers for binary strings are geometric crossovers under Hamming distance: they all produce offspring on the shortest path between parent binary strings in the Hamming space. Differently from binary strings where a single, natural definition of distance is normally used, for permutations many notions of edit distance are equally natural (based on swaps, adjacent swaps, insertions, reversals, transpositions and other edit moves). This leads to a number of natural notions of geometric crossover for permutations. Geometric crossovers for permutations are intimately connected with the notion of sorting algorithm: offspring are on the shortest path between parents, hence they are on the minimal sorting trajectory between parents using a specific edit move. Interestingly, this allows to implement geometric crossovers for permutations by using traditional sorting algorithms such as bubble sort, selection sort and insertion sort. Many pre-existing recombination operators for permutations are geometric crossovers. For example, PMX (partially matched crossover) and cycle crossover are geometric under swap distance. Interestingly, cycle crossover is geometric also under Hamming distance restricted to permutations.

Permutations with repetitions are a natural generalization of simple permutations in which each element is allowed to occur more than once. In this paper we start studying the application of geometric crossover to permutations with repetitions. In particular, we propose a new geometric crossover for permutations with repetitions that is a natural generalization of cycle crossover.

Grouping problems are interesting and NP-hard [6]. When applying evolutionary algorithms to grouping problems, the standard solution encoding is highly redundant. This affects badly the performance of traditional crossover. In previous work [16], we developed a geometric crossover for the graph partitioning problem based on a labeling-independent distance that filters the inherent redundancy of the solution encoding that performed very well.

A second difficulty with grouping problems is that traditional recombination does not preserve feasibility of offspring: recombining parents with the same grouping structures does not lead in general to offspring with the same structure, requiring a repairing mechanism to be applied to the offspring. Per se, the repairing mechanism is not necessarily negative as it can be interpreted as a form of mutation. However, when the extent of change in the offspring is not small, the repairing mechanism degenerates into macro-mutation with deleterious effect on performance.

In general, a much preferred way to deal with this problem is to design a recombination operator that naturally transmits parent feasibility to offspring. In this paper we show that the new cycle crossover for permutation with repetitions

naturally applies to grouping problems allowing to search only the space of feasible solutions without the need of any repairing mechanism. We then combine cycle crossover and labeling-independent crossover obtaining a new geometric crossover with both advantages that suits very well grouping problems with redundant encoding. We tested experimentally the new geometric crossovers on graph partitioning and report remarkable performance improvement.

The reminder of this paper is organized as follows. In Section 2, we introduce the geometric framework and review previous work on permutations. In Section 3, we introduce a new geometric crossover, extending the cycle crossover to permutations with repetitions, that preserves the sizes of repetition classes. In Section 4, we introduce the multiway graph partitioning problem and a previous geometric crossover based on the labeling-independent distance. In Section 5, we recast the graph partitioning problem in terms of permutations with repetitions and motivate the use of cycle crossover. We then combine cycle crossover and labeling-independent crossover into a new geometric crossover with both characteristics. In Section 6, we present experimental setting and results, and we draw conclusions in Section 7.

2 Geometric Framework

In this section, we report the essential concepts behind a theoretical framework of recent introduction that allows to analyze and design new crossover operators for any solution representation tailored to the problem at hand [20].

2.1 Geometric Preliminaries

The term *distance* or *metric* denotes any real valued function that conforms to the axioms of identity, symmetry and triangular inequality. A simple connected graph is naturally associated to a metric space via its *path metric*: the distance between two vertices in the graph is the length of a shortest path between the vertices. Given a set of editing operations (edit moves) well-defined over a set of syntactic objects, the *edit distance* between two syntactic objects is the minimum number of edit moves needed to transform one into the other. When the edit moves are reversible and every object can be transformed into any other using the edit moves available, the edit distance is a metric.

In a metric space (S, d), a *closed ball* is the set of the form $B(x; \delta) = \{y \in S \mid d(x, y) \leq \delta\}$ where $x \in S$ and δ is a positive real number called the radius of the ball. A *line segment* (or closed interval) is the set of the form $[x, y]_d = \{z \in S \mid d(x, z) + d(z, y) = d(x, y)\}$ where $x, y \in S$ are called extremes of the segment. Metric ball and metric segment generalize the familiar notions of ball and segment in the Euclidean space to any metric space through distance redefinition. These generalized objects look quite different under different metrics. Notice that a metric segment does not coincide to a shortest path connecting its extremes (*geodesic*) as in an Euclidean space. In general, there

may be more than one geodesic connecting two extremes; the metric segment is the union of all geodesics.

We assign a structure to the solution *set* by endowing it with a notion of distance d. $M = (S, d)$ is therefore a solution *space* and $L = (M, g)$ is the corresponding fitness landscape, where g is the fitness function over S.

2.2 Definition of Geometric Crossover

The following definition is *representation-independent* whereby crossover is well-defined for any representation. It is only *function of the metric* d associated with the search space being based on the notion of metric segment.

Definition 1 (Geometric crossover) *A binary operator GX is a geometric crossover under the metric d if all offspring are in the segment between their parents, i.e., $\forall x, y : GX(x, y) \in [x, y]_d$.*

A number of general properties for geometric crossover and mutation have been derived in [20]. The traditional crossover for K-ary vectors with n crossover points is geometric under the Hamming distance [20].

2.3 Geometric Crossover for Permutations

In previous work we have studied various crossovers for permutations, revealing that PMX [7], a well-known crossover for permutations, is geometric under swap distance. Also, we found that cycle crossover [24], another traditional crossover for permutations, is geometric under swap distance and under Hamming distance (geometricity under Hamming distance for permutations implies geometricity under swap distance but not vice versa). Finally, we showed that geometric crossovers for permutations based on edit moves are naturally associated with sorting algorithms: picking offspring on a minimum path between two parents corresponds to picking partially sorted permutations on the minimal sorting trajectory between the parents.

3 Cycle Crossover for Permutations with Repetitions

In Section 3.1 we introduce the notions of repetition class of a permutation with repetitions and class-preserving geometric crossover. In Section 3.2 we describe a generalization of cycle crossover for permutations with repetitions that is a class-preserving geometric crossover under Hamming distance and in Section 3.3 we point out its properties. Cycle crossover for simple permutations is geometric under swap distance. In Section 3.4 we show that the extension of cycle crossover to permutations with repetitions is not geometric under swap distance. This comes as a surprise because when the extended cycle crossover is applied to simple permutations it behaves exactly as the simple cycle crossover, hence it becomes geometric under swap distance. We elicit the origin of this counterintuitive result.

3.1 Geometric Crossover for Permutations with Repetitions

Permutations with Repetitions: In a permutation every elements occurs exactly once, e.g., (21453). In a permutation with repetitions the same value may occur more than once, e.g., (214154232), where 1 occurs twice, 2 occurs 3 times, 3 occurs once, 4 twice and 5 once. The numbers of repetitions of each element define the repetition class of the permutation. Two permutations with repetitions in which elements have the same number of repetitions belong to the same repetition class. All simple permutations (without repetitions) belong to the same repetition class.

Class-preserving Neighborhood Structure: Let us consider the set P of all the permutations with repetitions such as the element 1 is repeated n_1 times, element 2 is repeated n_2 times, and so on. So n_i is the size of group i. In P, any permutation has a fix number of repetitions for each group. So, all permutations with repetitions in P belong to the same repetition class.

We can define a neighborhood structure on P as follows: the neighbor is generated by swapping the positions of any two different element in the permutation. For example, swapping positions of the elements at positions 2 and 5 in 123231 gives 133221, so they are neighbors.

This operation transforms a permutation in P into another permutation in P: it does not change the sizes of the groups. In other words, the swap operation is a *class-preserving transformation*.

It is also easy to see that this neighborhood is symmetric (if p_1 is in the neighborhood of p_2, then p_2 is in the neighborhood of p_1) and connected (it exists a finite number of swaps that transform any permutation in P into any other permutation in P).

Notice that many other edit operations for simple permutations can be naturally extended to the case of permutations with repetitions and are indeed class-preserving transformations. For example, the block-reversal move that reverses a block of consecutive elements in the permutation does not change the class of permutation either. In this paper we focus on the case of the swap move.

Class-preserving Swap Distance: Passing from the neighborhood structure (symmetric and connected) to the corresponding notion of distance is straightforward: the distance between two solutions is simply the length of a shortest path connecting them in the neighborhood structure. This means that the distance between two permutations with repetitions (of the same repetition class) is the minimum number of swap operations required to transform one into the other. This is an edit distance hence it is a metric.

Class-preserving Geometric Crossover: By definition of geometric crossover, once we have a distance over a space we can define a crossover for that space. The geometric crossover has to pick offspring on a shortest path connecting two solutions. From previous work on geometric crossover for permutations, the extension to permutations with repetitions *seems* straightforward. It is immediate to implement geometric crossover for simple permutations by noticing that picking offspring on a shortest path means picking offspring on a minimal sorting trajectory to sort one permutation to the order of the other

by using (in our case) the swap move. This can be easily implemented by using a classical sort algorithm (selection sort in this case), applying a number of sorting moves and interrupting the sorting at a random point (before the sort is complete) and return the permutation partially sorted as offspring. The sort algorithm can also be used to measure the distance between two permutations. Selection sort can be used as a base for geometric crossover because it provably sorts simple permutations within the minimum number of swaps.

However, when applied to permutations with repetitions, selection sort is not guaranteed to sort using the minimum number of swaps. This is very counterintuitive; we explain why this happens in Section 3.4. Therefore, partially sorted permutations are not necessarily on the minimal sorting trajectory (shortest path) between parent permutations and selection sort cannot be used to implement exactly a geometric crossover under swap distance. However a recombination based on selection sort is class-preserving and it produces a very good approximation[4] of geometric crossover under swap distance, hence it can be thought to be a geometric crossover plus a small mutation.

An alternative way to build geometric crossovers for permutations with repetitions under swap distance is to generalize PMX and cycle crossover that are geometric under swap distance for simple permutations. Despite their looking, these operators are in fact sorting crossovers: they pick offspring on a minimal sorting trajectory between parents under swap move. The difference is that instead of doing this move by move (like sorting algorithms), they do it in one shot using a syntactic transformation equivalent to the sequential composition of a number of simple swaps. So, the above discussion about the non-geometricity of sorting crossover for permutations with repetitions applies to PMX and cycle crossover too. However cycle crossover is geometric also under Hamming distance. This allows us to generalize it to the case of permutations with repetitions and preserve geometricity under Hamming distance.

3.2 Generalization of Cycle Crossover

The extension of the cycle crossover we propose produces offspring of the same repetition class of the parents. This crossover has two phases: (i) finding cycles and (ii) mixing cycles. We explain these below by means of an example.

PHASE I (FINDING CYCLES):
Let us consider two parents of the same repetition class ($1 \times 2, 2 \times 2$ and 3×2) in Fig 1.a. In order to identify cycles, we proceed as follows.
(1) Pick a random position in parent A, e.g., position 5. In this position in parent A, we have the element 1 corresponding in parent B (at the same position) to the element 3: $1 \rightarrow 3$. Mark position 5 as taken.

[4] It is possible to formalize the notion of approximated geometric crossover rigorously. The approximation is the largest ratio between the length of the shortest path connecting parents *and* passing through the offspring over the length of the shortest path connecting parents. In a (perfect) geometric crossover, this ratio is 1. In an approximated geometric crossover this ratio is greater than 1. The larger the ratio, the worse the approximation.

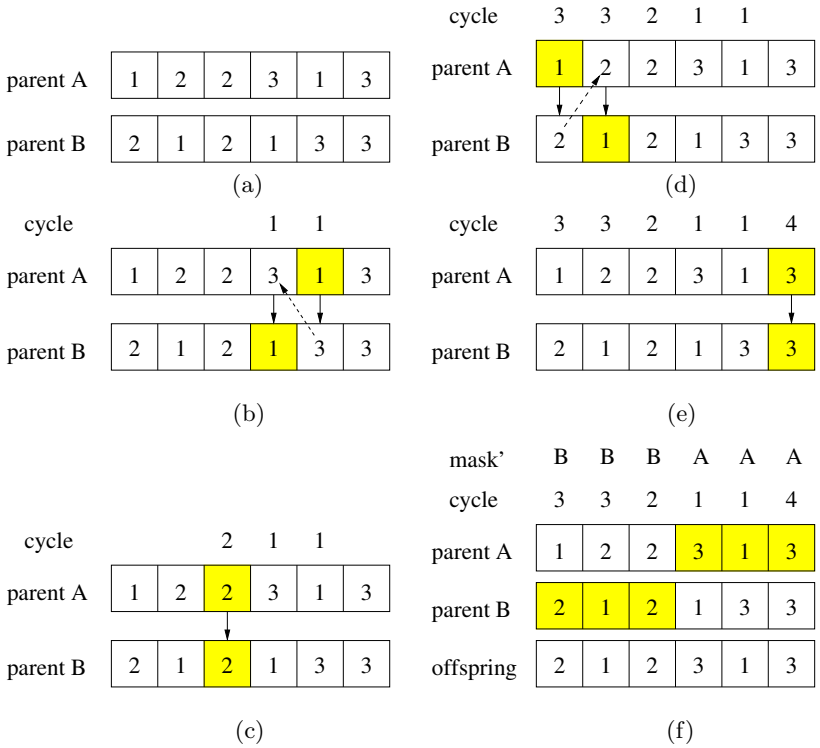

Fig. 1. Cycle crossover step by step

(2) Consider the corresponding element in parent B and pick at random any of its occurrence in parent A (among non-taken positions). In our example, pick any 3 in parent A, let us pick the one at position 4. The corresponding element in parent B is 1: $3 \rightarrow 1$. Mark position 4 as taken.

(3) Continue this procedure until you get an element of parent B that is the first element you considered in parent A. When this happens, we have found a cycle. In our example, the last element we got in parent B is 1 that is the same as the first element we considered in parent A. So we found the cycle: $1 \rightarrow 3, 3 \rightarrow 1$. This is shown in Fig. 1.b. Notice that the cycle involves the same number of repetitions in both parents. In our example, 1×1 and 3×1. Excluding the elements of cycle 1 from the two parents leave the remaining elements with the same number of repetitions in the two parents. So the "leftover" permutations are still of the same repetition class. In our example, the leftover repetition class is: 1×1, 2×2 and 3×1.

(4) Repeat loop (1)–(3) to find more cycles until all position have been marked with a cycle tag.

Continuing our example: (1′) Pick one free position in parent A at random. Say position 3. We have a 2. (2′) Corresponding element in parent B: $2 \rightarrow 2$. (3′) We already found a cycle: $2 \rightarrow 2$ (Fig. 1.c).

We then start searching for a third cycle: (1″) Pick one free position in parent A at random. Say position 1. We have a 1. (2″) Corresponding element in parent B: $1 \rightarrow 2$. Pick a 2 in parent A: the only one available is the one at position 2. Its corresponding element in parent B is: $2 \rightarrow 1$. (3″) We have found another cycle: $1 \rightarrow 2, 2 \rightarrow 1$ (Fig. 1.d).

We run the process one further time: (1‴) The only free position in parent A is position 6. We have 3. (2‴) The corresponding element in parent B is 3. (3‴) We have found a cycle: $3 \rightarrow 3$ (Fig. 1.e).

All the positions have been assigned to a cycle, so Phase I is over. Notice that the last iteration is always guaranteed to terminate with a cycle (and not with a simple sequence). The last position marked must be the end of the cycle.

PHASE II (MIXING CYCLES):
(1) Create a crossover mask with one entry for each cycle by randomly flipping a coin as many times as the number of cycles detected in the previous phase. In our example, we have 4 cycles and, say, the crossover mask we generate is `mask` = (ABBA). The entries in the mask indicate from which parent each cycle is inherited. In this example, the offspring will inherit the cycles 1 and 4 from parent A and the cycles 2 and 3 from parent B.
(2) We convert this "cycle" mask into a standard recombination mask by relabeling all the entries c_i in `cycle` as follows: $c_i \rightarrow \texttt{mask}(c_i)$ obtaining a new mask `mask'`.
(3) We perform standard mask-based crossover on the two parents using `mask'`, obtaining the offspring as shown in Fig. 1.f.

Notice that by construction every offspring has the same number of repetitions of the parents. This is because exchanging any cycle between parents is repetition-preserving.

3.3 Properties of Cycle Crossover

The new crossover has the following properties:

1. It is repetition class preserving.
2. It is a proper generalization of cycle crossover: when applied to simple permutations it behaves exactly like cycle crossover.
3. It is geometric under Hamming distance because at any position the element in the offspring equals the element at the same position of one of the parents.
4. This geometric crossover is defined over the induced sub-metric space obtained by restricting the original vector space endowed with Hamming distance to the space of permutations with repetition of the same repetition class. The latter space is much smaller than the former and it is, hence, quicker to search.
5. Applying this crossover to permutations with repetitions of different repetition class (with minor modifications), one obtains offspring with intermediate repetition class with respect to the repetition classes of the parents.

3.4 Non-geometricity under Swap Distance

Theorem 1. *Cycle crossover is not geometric under swap distance.*

Proof. We prove it by giving a counter-example. Let us consider two parents $A = (122313)$ and $B = (213132)$. We can have (at least) two cycles decompositions after Phase I:

cycle	1	1	2	3	3	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2

and

cycle	1	2	1	1	2	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2

By combining cycles of the first decomposition, one obtains offspring that are always in the segment between parents under swap distance. However, by combining cycles of the second decomposition, one obtains an offspring that is not in the segment between parents under swap distance:

mask'	A	B	A	A	B	B
cycle	1	2	1	1	2	2
parent A	1	2	2	3	1	3
parent B	2	1	3	1	3	2
offspring	1	1	2	3	3	2

$d_{sw}(A, B) = 3$: This can be seen from the first decomposition in cycles (minimal): 3 cycles of length 2 is equivalent to 3 swaps away.
$d_{sw}(A, \textit{offspring}) = 2$: By construction, the minimal decomposition the offspring and parent A in cycles is the cycle of length 3 taken from parent B. This means they are 2 swaps away.
$d_{sw}(\textit{offspring}, B) = 2$: By construction, the minimal decomposition the offspring and parent B in cycles is the cycle of length 3 taken from parent A. This means they are 2 swaps away.

Since the swap distance between parents is smaller then the sum of the swap distances between parents and offspring, cycle crossover is non-geometric under swap distance. This ends the proof. ■

It can be shown in general that the decomposition with the most cycles produces always offspring in the segment between parents under swap distance.

All the other decompositions lead to at least one offspring not in the segment between parents. So, *the cycle crossover restricted to the decomposition with the most cycles is geometric crossover under swap distance.*

Cycle crossover for simple permutations produces always offspring within the segment between parents under swap distance. Simple permutations can be thought as permutations with one repetition for each element. How then the result of geometricity under swap distance for simple permutations and non-geometricity under swap distance for permutations with repetitions connect? We know from abstract algebra that there is a unique decomposition in cycles for simple permutations. A unique decomposition is also the one with the most cycles. Hence in the special case of simple permutations, all the offspring are within the segment between parents.

For simple permutations, cycle crossover is geometric under swap distance and Hamming distance. More generally it can be shown that any geometric crossover for permutations that is geometric under Hamming distance is necessarily geometric under swap distance, but not vice versa[5] This is because the segment under Hamming distance is a subset of the segment under swap distance for any choice of extremes. When considering the more general case of permutations with repetitions, cycle crossover is geometric under Hamming distance but it is not geometric under swap distance. As a consequence, the implication geometricity under Hamming distance implies geometricity under swap distance ceases to hold true when stretched to permutations with repetitions.

With similar arguments as for cycle crossover, it can be shown that PMX and all sorting crossovers under swap move are in general non-geometric crossovers.

4 Graph Partitioning and Labeling-Independent Crossover

4.1 Multiway Graph Partitioning

Graph partitioning is an important problem that arises in various fields of computer science, such as sparse matrix factorization, VLSI circuit placement, network partitioning, and so on. Good partitioning of a system not only significantly reduces the complexity involved in the design process, but can also improve the timing performance as well as its reliability [8].

Let $G = (V, E)$ be an unweighted undirected graph, where V is the set of vertices and E is the set of edges. *K-way partition* is a partitioning of the vertex set V into K disjoint subsets $\{C_1, C_2, \ldots, C_K\}$. A K-way partition is said to be *balanced* if the difference of cardinalities between the largest and the smallest subsets is at most one. The *cut size* of a partition is defined to be the number of edges with endpoints in different subsets of the partition. The *K-way partitioning problem* is the problem of finding K-way balanced partition with minimum cut size.

[5] For example, PMX is geometric under swap distance but is not geometric under Hamming distance.

Since the K-way partitioning problem is NP-hard [6], attempts to solve partitioning problems have focused on finding heuristics which yield approximate solutions in polynomial time. Among such methods, the Kernighan-Lin algorithm [12] and the Fiduccia-Mattheyses algorithm (FM) [5] are representative. They are local search heuristics for 2-way partitioning. There have been a number of algorithms for K-way partitioning [3] [4] [11] [25]. There have been also several methods using genetic algorithms [10] [13] [18].

4.2 Geometric Crossover for Unlabeled Partitions

The standard representation of a solution for K-way graph partitioning is a vector r of size $|V|$ such as $r_i = j \Rightarrow v_i \in C_j$. Since the specific mapping of indices to partitions does not change how the graph is partitioned, each solution has $K!$ representations. Traditional crossover, that is geometric under Hamming distance, does not perform well on redundant encodings. In fact, for this encoding, the Hamming distance between two solutions is unnatural because it depends on the specific mapping between indices and partitions that is completely arbitrary. We proposed a distance measure, the labeling-independent distance, that eliminates this dependency completely [15]. In [16], we proposed a new geometric crossover (LI-GX) for graph partitioning based on a labeling-independent distance associated to the Hamming distance that filters the redundancy of the encoding. We showed that LI-GX can be implemented efficiently by using the Hungarian method [17] to normalize the labeling on the second parent to that of the first parent and then applying traditional crossover. LI-GX can be thought as restricting the search to the space of unlabelled-partitions only. It outperforms by far the traditional crossover (H-GX) that searches the whole space of labelled-partitions.

5 Cycle Crossovers for Graph Partitioning

5.1 Searching Balanced Partitions

In the multiway graph partitioning problem, one needs to keep the sizes of the partitions balanced. So this is a constrained optimization problem, where the constraint is the balancedness. Among all solutions (balanced or not), the feasible ones are only those that are balanced. The way we have dealt with it in previous work [16] is searching using a crossover that searches the space of all solutions and then applying a repairing mechanism, that can be thought as a mutation, that repairs offspring and makes them feasible (balanced). There are other ways to deal with constraints. An alternative method that does not need to use any repairing mechanism is to have a geometric crossover that searches only the space of balanced solutions. This is the approach we take here. This reduces the size of the search space considerably (the set of balanced solutions is a fraction of the whole search space).

Representation: The starting point for restricting the search to balanced-partitions only is to see the object representing the solution not as a vector

of integer but as a permutation with repetitions. Every position in the permutation still represents a vertex of the graph and every integer still represents the label of the group the vertex at that position is assigned to.

A solution is balanced when all the partitions have approximatively the same number of vertices. This means that in the representation, there will be a similar number of repetitions of each element (integer). *Notice that two balanced solutions do not necessarily belong to the same repetition class.*

Equally Balanced Initial Population: In order to restrict the search only to the space of equally balanced partitions, we need to seed the initial population with solutions having *for the same partition exactly the same size for all solutions* (belonging to the same repetition class). Seeding the population with balanced solutions is not sufficient.

Balanced Crossover = Cycle Crossover: Cycle crossover preserves repetition class. Hence given two balanced parents belonging to the same repetition class, it returns offspring of the same repetition class, hence balanced. So there is no need for repairing mutations.

Balanced Mutation = Swap Mutation: We need to use a mutation that keeps a permutation with repetition within the same repetition class. So that, if a solution is balanced, the mutated solution is still balanced. A simple mutation with this characteristic is the simple swap mutation based on the swap move. Notice that the swap move is a good one for the graph partitioning problem because it produces a landscape with a smooth trend (solutions one swap away have very similar fitness). The swap move is also a good base for local search to search the space of balanced grouping only. This move can be used as a base of a more sophisticated mutation that decreases its probability exponentially with the distance from the parent solution.

The cycle crossover suggested is (almost) geometric under swap distance. Hence, our swap mutation and cycle crossover are defined over and search the same metric space. This can also be extended to the local search. Having different operators searching the same space is interesting because it is then possible to interpret their interactions in a simple geometric way.

5.2 Combining Labeling-Independent Crossover and Cycle Crossover

Combination of Relabeling and Balanced Solutions: Cycle crossover (Cycle H-GX) searches the space of balanced partitions. LI-GX (see Section 4) searches the space of labeling-independent partitions. We combine these two geometric crossovers obtaining a new geometric crossover with both advantages: it operates fully within the space of labeling-independent balanced partition space, which is a fraction of the original space and could produce highly competitive performance. *The new crossover (Cycle LI-GX) consists of a labeling-normalization phase before applying cycle crossover.* The normalization is done using the Hungarian method as for LI-GX [16].

Geometricity of Compound Crossover: Cycle LI-GX is still geometric on the phenotypic space restricted to balanced phenotypes. It is in fact the tradi-

tional mask-based crossover restricted to the subspace of vectors that take the form of fixed-size class permutations with repetitions.
Equally Balanced Solutions: Relabeling a solution does not affect its balancedness but it may change its repetition class. Cycle crossover without normalization is able to deal with different partition sizes. Cycle crossover plus normalization requires all partitions to have exactly the same size. In this special case, the relabeling does not change the repetition class of the solution and the cycle crossover is therefore applied to solutions of the same repetition class.
Inexact Balance and Cycle Crossover: However we would like to apply the cycle crossover with normalization not only to the restricted class of problems with partitions of exactly the same size but to all balanced partitions. We therefore have to further extend cycle crossover to the recombination of solutions belonging to different repetition classes. When attempting to use the cycle crossover on permutations of different repetition classes, we may not obtain a proper decomposition in cycles. We solved this problem heuristically and considered some non-cycles (paths) as cycles.

6 Experiments

6.1 Genetic Framework

We used the general structure of hybrid steady-state genetic algorithms. In the following, we describe the framework of genetic algorithm used in our experiments. Under this framework, we will change only the crossover operator.

- *Encoding*: We use a K-ary string for each chromosome to represent a K-way partition. For example, if vertex v_i belongs to partition C_j, the value of the i^{th} gene is j.
- *Initialization*: We randomly create p chromosomes. Each chromosome satisfies a balance criterion. We set the population size p to be 50.
- *Selection*: We use the roulette-wheel-based proportional selection scheme. The probability that the best chromosome is chosen was set to four times higher than the probability that the worst chromosome is chosen.
- *Mutation*: After cycle crossover (Cycle H-GX) or normalized cycle crossover (Cycle LI-GX), we run the following swap mutation.

```
function: offspring mutate(offspring) {
    for i = 1 to N {
        if(p_mut > rnd_real_range(0,1))
            offspring = do_rnd_swap(offspring, i);
    }
    return offspring;
}
```

where $N = |V|$, `rnd_real_range(0,1)` returns random real numbers in the range [0,1] and `do_rnd_swap(offspring, i)` chooses a random point j ($j \neq i$) and swaps positions i and j.

The mutation parameter `p_mut` is set to be 0.005. Then, the expected Hamming distance between chromosomes before and after mutation is approximately 1 percent of the problem size $|V|$.

- *Local optimization*: Sanchis [25] extended the FM algorithm for K-way partitioning. The algorithm considers all possible moves of each vertex from its home set to any of the others. He showed that this direct multiway partitioning approach obtained better solutions compared to the recursive approach for random networks. As local optimization engine in our genetic algorithm, we use its variation proposed in [13]. Its time complexity is $O(K|E|)$.
- *Replacement*: If it is superior to the closer parent, the offspring replaces the closer parent, and if not, the other parent is replaced if the offspring is better. Otherwise the worst in the population is replaced.
- *Stopping criterion*: For stopping, we use the number of consecutive fails to replace one of the parents. We set the number to be 50.

6.2 Test Environment

Before showing the experimental results, we first introduce the benchmarks used in this experiment and test environment. We tested on a total of eight graphs which consist of two groups of graphs. They are composed of eight graphs with 500 vertices from [9] (four random graphs G*.* and four random geometric graphs U*.*). The two classes were used in a number of other graph-partitioning studies [1] [2] [14] [19]. More detailed description of them is given in [14].

We conducted tests on 32-way and 128-way partitioning. A *C* language program was used on a Pentium III 1GHz computer with Linux operating system. It was compiled using *gcc* compiler.

Although the instances of the test-bed are from standard library, most research have focused on 2-way partitioning (bi-partitioning). Moreover, many research about multiway partitioning dealt with circuit partitioning (hyper-graph partitioning). However we know the lower bounds for 32-way partitioning instances from previous literature [10] [13] [16].

6.3 Results

We compare the geometric crossover based on the Hamming distance (5pt H-GX), the geometric crossover based on the corresponding labeling-independent distance (5pt LI-GX), the geometric crossover based on the Hamming distance restricted to permutation with repetitions (Cycle H-GX), and the geometric crossover based on the corresponding labeling-independent distance (Cycle LI-GX). Notice that these crossovers search different search spaces:

crossover	search space
5pt H-GX	the space of all labeled partitions
5pt LI-GX	the space of all unlabeled partitions
Cycle H-GX	the space of all labeled well-balanced partitions
Cycle LI-GX	the space of all unlabeled well-balanced partitions

Table 1. The Results of 32-way Partitioning

Graph	Best	5pt H-GX			5pt LI-GX		
	Known	Best	Ave†	Gen(CPU‡)	Best	Ave†	Gen(CPU‡)
G500.2.5	178	182	185.18	1091(173.33)	178	181.77	1529(180.30)
G500.05	624	626	637.25	1424(330.64)	624	630.07	2367(334.80)
G500.10	1574	1576	1587.23	1984(713.07)	1573	1581.40	2422(571.50)
G500.20	4037	4040	4049.44	2247(1502.26)	4034	4044.89	2522(1245.96)
U500.05	113	112	120.65	1327(319.04)	112	116.75	1599(331.95)
U500.10	529	534	542.75	1163(449.76)	531	537.04	1494(483.50)
U500.20	1825	1837	1846.30	1123(814.94)	1832	1841.02	1353(747.04)
U500.40	5328	5363	5389.93	1043(1399.31)	5353	5380.30	1374(1398.19)

† Average over 100 runs.
‡ CPU seconds on Pentium III 1GHz.

Table 2. The Results of 32-way Partitioning

Graph	Best	Cycle H-GX			Cycle LI-GX		
	Known	Best	Ave†	Gen(CPU‡)	Best	Ave†	Gen(CPU‡)
G500.2.5	178	177	185.59	1269(154.60)	177	180.50	1582(204.50)
G500.05	624	627	637.58	1660(282.69)	623	628.63	2232(378.46)
G500.10	1574	1575	1587.38	1987(537.10)	1573	1580.53	2381(641.58)
G500.20	4037	4039	4049.65	2198(1270.58)	4035	4043.29	2538(1354.42)
U500.05	113	113	120.41	1245(300.42)	109	112.60	2056(371.45)
U500.10	529	524	539.67	1263(472.37)	523	528.50	2086(583.02)
U500.20	1825	1834	1843.80	1170(790.02)	1825	1831.55	1646(844.36)
U500.40	5328	5372	5391.77	999(1314.44)	5348	5365.00	1691(1515.15)

† Average over 100 runs.
‡ CPU seconds on Pentium III 1GHz.

Table 1 and Table 2 show the results of 32-way partitioning. On random graphs, Cycle H-GX could not dominate 5pt H-GX on averages, but it performed better on the best. On random geometric graphs, Cycle H-GX performed better than 5pt H-GX both on averages and the best. Cycle LI-GX and 5pt LI-GX

Table 3. The Results of 128-way Partitioning

Graph	5pt H-GX			5pt LI-GX		
	Best	Ave†	Gen(CPU‡)	Best	Ave†	Gen(CPU‡)
G500.2.5	316	320.14	844(535.13)	310	314.08	950(759.72)
G500.05	850	853.09	869(688.03)	839	843.61	1020(947.68)
G500.10	1904	1907.78	932(1300.94)	1894	1898.03	1168(1316.21)
G500.20	4568	4571.93	965(2333.56)	4560	4566.40	1116(2261.33)
U500.05	697	704.06	935(910.83)	695	702.90	978(1227.62)
U500.10	1679	1684.13	913(1302.62)	1676	1683.47	921(1618.17)
U500.20	3836	3841.45	890(1437.07)	3836	3841.44	874(1790.69)
U500.40	8066	8068.54	853(1971.41)	8065	8068.77	831(2250.17)

† Average over 100 runs.
‡ CPU seconds on Pentium III 1GHz.

Table 4. The Results of 128-way Partitioning

Graph	Cycle H-GX			Cycle LI-GX		
	Best	Ave†	Gen(CPU‡)	Best	Ave†	Gen(CPU‡)
G500.2.5	311	314.54	911(319.23)	310	313.05	964(694.41)
G500.05	839	844.13	977(463.58)	840	843.19	1058(876.97)
G500.10	1896	1899.32	1033(766.90)	1893	1896.71	1191(1200.03)
G500.20	4564	4567.94	1004(1774.22)	4560	4565.06	1164(2149.44)
U500.05	695	701.40	941(684.25)	692	698.96	1219(1480.44)
U500.10	1673	1682.74	923(1095.62)	1675	1681.55	988(1656.25)
U500.20	3838	3840.37	864(1186.80)	3835	3841.15	887(1799.27)
U500.40	8065	8067.74	845(1493.73)	8065	8068.42	832(2115.29)

† Average over 100 runs.
‡ CPU seconds on Pentium III 1GHz.

always outperformed Cycle H-GX and 5pt H-GX. Cycle LI-GX showed more improved performance compared with 5pt LI-GX. Except on U500.40, it found lower bounds better than or equal to the best known.

Table 3 and Table 4 show the results of 128-way partitioning. Cycle LI-GX also performed best. Except on G500.05 and U500.10, it found the best solution among them. The performance of Cycle H-GX was better than in the case of 32-way partitioning. On random geometric graphs, it outperformed 5pt LI-GX. But on random graphs, 5pt LI-GX performed better. 5pt H-GX was always dominated by others.

In summary, we got visible improvement for all the tested instances. In particular, for 32-way partitioning on random geometric graphs, there was large improvement.

Now we explain the computing time. For small number K, normalization by the Hungarian method affects computational time little. In 32-way partitioning, Cycle LI-GX was about 1.2 times slower than Cycle H-GX. But, normalization time increases as K increases. In fact, Cycle LI-GX was about 1.7 times slower

than Cycle H-GX in 128-way partitioning. Cycle crossovers were faster than 5-point crossovers. In results, Cycle H-GX and Cycle LI-GX were faster than 5pt H-GX and 5pt LI-GX, respectively. Consequently, Cycle H-GX was fastest among them.

7 Conclusions

Geometric crossover is a representation-independent generalization of the traditional crossover defined using the distance of the solution space. By choosing a distance firmly rooted in the syntax of the solution representation and tailored to the problem at hand as basis for geometric crossover, one can design good new crossovers for any representation and any problem.

Using this framework, in this paper, we have designed a new crossover for permutations with repetitions that naturally suits partition problems. We have then combined this crossover with another geometric crossover that we had developed in previous work on the graph partitioning problem obtaining a new, much superior geometric crossover that suits partition problems with redundant encodings. In extensive experimentation, we had demonstrated that this crossover outperforms previously known methods, either providing new lower bounds or equalling known best lower bounds in a variety of graph partitioning benchmark problems.

References

1. R. Battiti and A. Bertossi. Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Transactions on Computers*, 48(4):361–385, 1999.
2. T. N. Bui and B. R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 1996.
3. J. Cong and S. K. Lim. Multiway partitioning with pairwise movement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 512–516, 1998.
4. J. Cong, S. K. Lim, and C. Wu. Performance driven multi-level and multiway partitioning with retiming. In *Proceedings of the ACM/IEEE-CAS/EDAC Design Automation Conference*, pages 274–279, 2000.
5. C. Fiduccia and R. Mattheyses. A linear time heuristics for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.
6. M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, 1979.
7. D. E. Goldberg and R. Lingle. Alleles, loci, and the travelling salesman problem. In *Proceedings of the First International Reference on Genetic Algorithms and their Applications*, pages 154–159, 1985.
8. T. C. Hu and E. S. Kuh. *VLSI Curcuit Layout Theory and Design*. IEEE Press, New York, 1985.
9. D. S. Johnson, C. Aragon, L. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation, Part 1, graph partitioning. *Operations Research*, 37:865–892, 1989.

10. S. J. Kang and B. R. Moon. A hybrid genetic algorithm for multiway graph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 159–166, 2000.
11. G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.
12. B. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49:291–307, Feb. 1970.
13. J. P. Kim and B. R. Moon. A hybrid genetic search for multi-way graph partitioning based on direct partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 408–415, 2001.
14. Y. H. Kim and B. R. Moon. Lock-gain based graph partitioning. *Journal of Heuristics*, 10(1):37–57, January 2004.
15. Y. H. Kim and B. R. Moon. New topologies for genetic search space. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1393–1399, 2005.
16. Y. H. Kim, Y. Yoon, A. Moraglio, and B. R. Moon. Geometric crossover for multiway graph partitioning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1217–1224, 2006.
17. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.
18. G. Laszewski. Intelligent structural operators for the k-way graph partitioning problem. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 45–52, July 1991.
19. P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91, 2000.
20. A. Moraglio and R. Poli. Topological interpretation of crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1377–1388, 2004.
21. A. Moraglio and R. Poli. Geometric crossover for the permutation representation. *Technical Report CSM-429*, 2005.
22. A. Moraglio and R. Poli. Topological crossover for the permutation representation. In *GECCO 2005 Workshop on Theory of Representations*, 2005.
23. A. Moraglio, R. Poli, and R. Seehuus. Geometric crossover for biological sequences. In *Proceedings of European Conference on Genetic Programming*, pages 121–132, 2006.
24. I. M. Oliver, D. J. Smith, and J. R. C. Holland. A study of permutation crossover operators on the travelling salesman problem. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 224–230, 1987.
25. L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, 1989.