2006 IEEE Congress on Evolutionary Computation
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Product Geometric Crossover for the Sudoku Puzzle

Alberto Moraglio, Julian Togelius and Simon Lucas

Abstract— **Geometric crossover is a representation-independent definition of crossover based on the distance of the search space interpreted as a metric space. It generalizes the traditional crossover for binary strings and other important recombination operators for the most used representations. Using a distance tailored to the problem at hand, the abstract definition of crossover can be used to design new problem specific crossovers that embed problem knowledge in the search. In recent work, we have introduced the important notion of product geometric crossover that enables the construction of new geometric crossovers combining pre-existing geometric crossovers in a simple way. In this paper, we use it to design an evolutionary algorithm to solve the Sudoku puzzle. The different types of constraints make Sudoku an interesting study case for crossover design. We conducted extensive experimental testing and found that, on medium and hard problems, the new geometric crossovers perform significantly better than hill-climbers and mutations alone.**

I. Introduction

Geometric crossover [6] is a representation-independent recombination operator defined over the distance of the search space. Informally, geometric crossover requires the offspring to lie between parents.

Geometric crossover generalizes many pre-existing recombination operators for the major representations used in evolutionary algorithms, such as binary strings [6], real vectors [6], permutations [7] [9], syntactic trees [8] and sequences [10].

They are defined in geometric terms using the notions of line segment and ball. These notions and the corresponding genetic operators are well-defined once a notion of distance in the search space is well-defined. This way of defining search operators as function of the search space is opposite to the standard way [3] in which the search space is seen as a function of the search operators employed. This viewpoint greatly simplifies the relationship between search operators and fitness landscape and has allowed us to give simple *rule-of-thumbs* to build/choose crossover operators for the problem at hand that are likely to perform well.

Using the abstract definition of geometric crossover and choosing a distance firmly rooted in the syntax of the solution representation as the basis for geometric crossover, one can derive new crossovers for any representation.

So far we have proven that a number of important pre-existing recombination operators for the most used representations are geometric crossovers (geometricity by analysis). We have also applied the abstract definition of geometric crossover to distances firmly rooted in a specific solution representation and designed brand-new crossovers (geometricity by construction). An appealing way to build new geometric crossovers is starting from recombination operators that are known to be geometric and derive new geometric crossovers by *geometricity-preserving transformations/combinations* that when applied to geometric crossovers return geometric crossovers (geometricity by derivation).

Alberto Moraglio is with the Department of Computer Science, University of Essex, UK (email: amoragn@essex.ac.uk).

Julian Togelius is with the Department of Computer Science, University of Essex, UK (email: jtogel@essex.ac.uk).

Simon Lucas is with the Department of Computer Science, University of Essex, UK (email: sml@essex.ac.uk).

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Fig. 1. Example of Sudoku puzzle.

The cartesian product of geometric crossover allows the construction of new geometric crossovers combining preexisting geometric crossovers in a very simple way. Given a number of representations and relative geometric crossovers, the geometric crossover for the compound representation taking the shape of a vector of a number of basic representations is the compound recombination operator that applies to each representation in the vector its relative geometric crossover. The distance associated with the product crossover is simply the sum of the distances of the composing crossovers.

Sudoku is a logic-based placement puzzle. The aim of the puzzle is to enter a digit from 1 through 9 in each cell of a 9x9 grid made up of 3x3 subgrids (called "regions"), starting with various digits given in some cells (the "givens"). Each row, column, and region must contain only one instance of each digit. In fig 1 we show an example Sudoku puzzle. Sudoku puzzles with a unique solution are called proper sudoku, and the majority of published grids are of this type.

Published puzzles often are ranked in terms of difficulty. Perhaps surprisingly, the number of givens has little or no bearing on a puzzle's difficulty. It is based on the relevance and the positioning of the numbers rather than the quantity of the numbers.

The 9x9 Sudoku puzzle of any difficulty can be solved very quickly by a computer. The simplest way is to use some brute force trial-and-error search employing back-tracking. Constraint programming is a more efficient method that ap-

0-7803-9487-9/06/$20.00/©2006 IEEE

plies the constraints successively to narrow down the solution space until a solution is found or until alternate values cannot otherwise be excluded, in which case backtracking is applied. A highly efficient way of solving such constraint problems is the Dancing Links Algorithm, by Donald Knuth [4].

The general problem of solving Sudoku puzzles on $n^2 \times n^2$ boards of n x n blocks is known to be NP-complete [12]. This means that, unless P=NP, the exact solution methods that solve very quickly the 9x9 boards take exponential time in the board size in the worst case. However, it is unknown weather the general Sudoku problem restricted to puzzles with unique solutions remains NP-complete or becomes polynomial.

Solving Sudoku puzzles can be expressed as a graph coloring problem. The aim of the puzzle in its standard form is to construct a proper 9-coloring of a particular graph, given a partial 9-coloring.

A valid Sudoku solution grid is also a Latin square. Sudoku imposes the additional regional constraint. Latin square completion is known to be NP-complete. A further relaxation of the problem allowing repetitions on columns (or rows) makes it polynomially solvable.

Admittedly evolutionary algorithms are not the best technique to solve Sudoku because they do not exploit systematically the problem constraints to narrow down the search. However, Sudoku is an interesting study case because it is a relatively simple problem but not trivial since is NP-complete, and the different types of constraints make Sudoku an interesting playground for crossover design. In particular, its structure makes it a perfect candidate for product geometric crossover.

In this paper we use the geometric framework and its extension to the cartesian product of geometric crossover, to design an evolutionary algorithm to solve the Sudoku puzzle. We report extensive experimental results.

The paper is organized as follows. In section 2 we present the geometric framework and its extension to product geometric crossover. In section 3, we design new geometric crossovers tailored to Sudoku using the product geometric crossover. In section 4, we introduce a new heuristic logic-based mutation built for Sudoku. In section 5, we test experimentally our new search operators. In section 6, we draw conclusions.

II. Geometric framework

A. Geometric preliminaries

In the following we give necessary preliminary geometric definitions. The following definitions are taken from [1].

The terms *distance* and *metric* denote 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 nodes in the graph is the length of a shortest path between the nodes. Similarly, an edge-weighted graph with strictly positive weights is naturally associated to a metric space via a *weighted path metric*.

In a metric space (S, d) a *closed ball* is the set of the form $B(x; r) = \{y \in S | d(x, y) \leq r\}$ 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 $[x; y] = \{z \in S | d(x, z) + d(z, y) = d(x, y)\}$ 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. Notice that d is arbitrary and need not have any particular connection or affinity with the search problem at hand.

B. Geometric crossover definition

The following definitions are *representation-independent* therefore 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: (Image set) The *image set* $Im[OP]$ of a genetic operator OP is the set of all possible offspring produced by OP with non-zero probability.

Definition 2: (Geometric crossover) A binary operator is a geometric crossover under the metric d if all offspring are in the segment between its parents.

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

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

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

A number of general properties for geometric crossover and mutation have been derived in [6].

Result: traditional crossover is geometric under Hamming distance.

C. Geometric crossover for permutations

In previous work we have studied various crossovers for permutations, revealing that PMX [2], a well-known crossover for permutations, is geometric under swap distance. Also, we found that Cycle crossover [2], 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.

D. Geometric crossover landscape

Geometric operators are defined as functions of the distance associated with the search space. However, the search space does not come with the problem itself. The problem consists only of a fitness function to optimize, that defines what a solution is and how to evaluate it, but it does not give any structure on the solution set. The act of putting a structure over the solution set is part of the search algorithm design and it is a designer's choice. A fitness landscape is the fitness function plus a structure over the solution space. So, for each problem, there is one fitness function but as many fitness landscapes as the number of possible different structures over the solution set. In principle, the designer could choose the structure to assign to the solution set completely independently from the problem at hand. However, because the search operators are defined over such a structure, doing so would make them decoupled from the problem at hand, hence turning the search into something very close to random search.

In order to avoid this one can exploit problem knowledge in the search. This can be achieved by carefully designing the connectivity structure of the fitness landscape. For example, one can study the objective function of the problem and select a neighbourhood structure that couples the distance between solutions and their fitness values. Once this is done problem knowledge can be exploited by search operators to perform better than random search, even if the search operators are problem-independent (as in the case of geometric crossover and mutation).

Under which conditions is a landscape well-searchable by geometric operators? As a rule of thumb, geometric mutation and geometric crossover work well on landscapes where the closer pairs of solutions, the more correlated their fitness values. Of course this is no surprise: the importance of landscape smoothness has been advocated in many different contexts and has been confirmed in uncountable empirical studies with many neighborhood search meta-heuristics [11].

Rule-of-thumb 1: if we have a good distance for the problem at hand than we have good geometric mutation and good geometric crossover

Rule-of-thumb 2: a good distance for the problem at hand is a distance that makes the landscape "smooth"

E. Product geometric crossover

In recent work [5] we have introduced the notion of product geometric crossover.

Theorem 1: Cartesian product of geometric crossover is geometric under the sum of distances

This theorem is very interesting because it allows one to build new geometric crossovers by combining crossovers that are known to be geometric. In particular, this applies to crossovers for mixed representations. The compounding geometric crossovers do not need to be independent, for the cartesian crossover to be geometric:

- *Multi-crossover*: same representation, same crossover n times
- *Hybrid crossover*: same representation, different crossover for each projection
- *Hybrid representation crossover*: different representation for each projection (and different crossover)
- *No independence required*: composing geometric crossovers do not need to be independent

III. Geometric design for Sudoku

A. Sudoku constraints, search spaces and genetic operators

Sudoku is a constraint satisfaction problem with 4 types of constraints:

1) Fixed elements
2) Rows are permutations
3) Columns are permutations
4) Boxes are permutations

It can be cast as an optimization problem by choosing some of the constraints as hard constraints that all solutions have to respect, and the remaining constraints as soft constraints that can be only partially fulfilled and the level of fulfillment is the fitness of the solution. Notice that there is more than one combination available to set which are hard constraints and which are soft constraints. We have considered two search spaces, the first respecting one constraint, the other respecting two constraints at the same time. Trying to further restrict the search space, requiring to respect three constraints at the same time seems not to be feasible: even only generating a random solution within the space of grids respecting the first three constraints is a NP-complete problem being equivalent to finding a solution to a Latin square.

1) Restricted Hamming space:

- *Hard constraints*: fixed positions
- *Soft constraints*: permutations on rows, columns and boxes
- *Distance*: Hamming distance between grids
- *Feasible geometric mutation*: change any non-fixed elements
- *Feasible geometric crossover*: traditional crossover over the vector obtained by joining the rows of the grid.

It is easy to see that these mutation and crossover preserve fixed positions from parent grids to offspring grids. The mutation is 1-geometric under Hamming distance. The crossover, being the traditional crossover, is geometric under Hamming distance.

To restrict the search to the space of grids with fixed positions, the initial population must be seeded with feasible random solutions taken from the feasible space, which is, by solutions respecting the fixed position constraints.

2) Row-swap space:

- *Hard constraints*: fixed positions and permutations on rows
- *Soft constraints*: permutations on columns and boxes
- *Distance*: sum of swap distances between paired rows (row-swap distance)

- *Feasible geometric mutation*: swap two non-fixed elements in a row
- *Feasible geometric crossover*: row-wise PMX and row-wise cycle crossover

This mutation preserves both fixed positions and permutations on rows because swapping elements within a row that is a permutation returns a permutation. The mutation is 1-geometric under row-swap distance.

Row-wise PMX and row-wise cycle crossover recombine parent grids applying respectively PMX and cycle crossover to each pair of corresponding rows. In case of PMX the crossover points can be selected to be the same for all rows, or be random for each row. In terms of offspring that can be generated, the second version of row-wise PMX includes all the offspring of the first version.

Simple PMX and simple cycle crossover applied to parent permutations return always permutations. They also preserve fixed positions. This is because both are geometric under swap distance and in order to generate offspring on a minimal sorting path between parents using swaps (sorting one parent into the order of the other parent) they have to avoid swaps that change common elements in both parents (elements that are already sorted). Therefore also row-wise PMX and row-wise cycle crossover preserve both hard constraints.

Using the product geometric crossover theorem, it is immediate that both row-wise PMX and row-wise cycle crossover are geometric under row-swap distance, since simple PMX and simple cycle crossover are geometric under swap distance. Since simple cycle crossover is also geometric under Hamming distance (restricted to permutations), row-wise cycle crossover is also geometric under Hamming distance.

To restrict the search to the space of grids with fixed positions and permutations on rows, the initial population must be seeded with feasible random solutions taken from this space. Generating such solutions can be done still very efficiently.

The row-swap space presents two advantages to the restricted Hamming space: (i) the search space of feasible solutions is much smaller and (ii) this restriction includes the optimum grid and prunes only grids with lower fitness.

B. Sudoku fitness landscapes

One important benefit of knowing that a certain search operator is geometric under a specific distance is that it is possible to tell a priori whether the search operator is a good choice for a specific problem. From previous work, we know that, as a rule-of-thumb, search operators associated to a fitness landscape with a smooth trend, in which closer solutions have stronger fitness correlation, are likely to perform well. Although this is quite intuitive for mutation operators, the geometric framework allows the extension to the case of crossover operators. In the following, we will show that the fitness landscapes associated with both spaces introduced in the previous section are smooth, making the search operators proposed a good choice for Sudoku.

Fitness function (to maximize): sum of number of unique elements in each row, plus, sum of number of unique elements in each column, plus, sum of number of unique elements in each box. So, for a 9×9 grid we have a maximum fitness of $9 \cdot 9 + 9 \cdot 9 + 9 \cdot 9 = 243$ for a completely correct Sudoku grid and a minimum fitness little more than $9 \cdot 1 + 9 \cdot 1 + 9 \cdot 1 = 27$ because for each row, column and square there is at least one unique element type. The maximum global fitness variation is therefore little less than $\Delta F = 216$.

Under Hamming distance, a single element change affects the current fitness of -1 0 +1 for the row, for the column and for the box the elements belong to. So the total change in fitness due to a single element change is between -3 and +3. The maximum local fitness variation is $\Delta f_H = 3$.

Under row-swap distance, a single swap in a row affects the current fitness of 0 for the row (permutation is preserved), in the range between -2 and +2 for the columns touched (a single swap can fix or unfix at most one element in two columns at the same time) and between -2 and +2 for the boxes touched. So the total change in fitness due to a single swap in a row is between -4 and 4. The maximum local fitness variation is $\Delta f_{rs} = 4$.

A simple measure of smoothness of the landscape is maximum local variation over maximum global variation. The smaller this measure, the smoother the landscape. $S_H = 3/216$ and $S_{rs} = 4/216$. *The fitness landscapes are both very smooth* when compared with the two extreme of smoothness 0 (no change between neighbor solutions) and 1 (maximum and minimum are neighbors).

Interestingly, the Hamming space is smoother than the row-swap space. So, in terms of smoothness the Hamming space is a better choice than the row-swap space.

It is also interesting to notice that the maximum global fitness variation increases (linearly) with the grid size, whereas the the maximum local fitness variation is constant with the grid size. *So, for increasing size of the problem, the fitness landscape becomes smoother.* So this may counteract the increasing difficulty of the search due to a larger space.

IV. SMART-SQUARE MUTATION

Since we want to get to the optimum as much as possible, and evolutionary algorithms are lazy optimiser, we use a heuristic mutation built for Sudoku.

The smart square mutation applies the most obvious constraints to the possible values of a square.

For example, if a column has a fixed '9' in it, then all the squares in that column have '9' removed from the set of possible values that it could take.

These constraints are applied for rows, columns, and subsquares (regions). For most sudoku grids, the application of these constraints from the initial set of fixed numbers will fix some further numbers i.e. certain squares will have only one possible value. Such squares are added to the set of fixed numbers, and the constraints thus arising are propagated. This process is iterated until no further numbers can be fixed. When the process is finished, each mutation now selects a

square from the set of ones that are still free, and chooses only among the feasible values for that square.

We implemented this as an obvious and irresistible improvement over a simplistic mutation operator that did not account for these constraints. However, there are many further sudoku constraints that we could have implemented, and taken to their logical extreme would solve most sudoku problems without the need for any search (evolutionary or otherwise).

V. Experiments

A. Genetic Framework

We tested the various operators with a steady state evolutionary algorithm. In the following, we describe the framework used in our experiments. The parameters were subject to extensive automatic tuning.

- *Encoding*:
  Each individual was encoded as an array of 81 integers in the range [0..9], which was interpreted as a vector of 9 rows.
- *Initialization*:
  For the hamming space experiments, each candidate solution of the first generation had the contents all non-fixed positions set to random integers in the range [1..9]. For the swap space experiments, each row of each candidate solution in the first generation was a random valid permutation of all numbers in the range[1..9], respecting fixed positions. See section III for more information.
- *Population*:
  The population size was set to 5000, with an elite of 2500 candidate solutions.
- *Selection and replacement*:
  Each generation, all candidate solutions were evaluated using the fitness function described in section III. The whole population was then sorted based on fitness, so that the 2500 best candidates formed the new elite. The other half of the population was replaced with 2500 new candidates, created by crossover and mutation (using the current operators) from two candidates randomly selected from the elite.
- *Crossover*:
  We used the crossover operators described in section III. All crossovers operate within the bounds of single rows (so e.g. PMX is row-wise PMX), except two-point crossover, which is traditional two-point crossover applied to the whole grid seen as a vector, and the whole-row crossover, which randomly selects whole rows from either of the two parents, and is geometric under both distances. Cycle crossover takes two forms: onecycle, the form proposed by Goldberg [2], which considers only one cycle, and multicycle, which considers all possible cycles. Uniform swap is a sorting crossover based on edit-swap-move.
- *Mutation*:
  For each newly formed candidate in a population, the current mutation operator was applied with probability 0.8. A number of mutation operators were used. For the hamming space, we used point mutation, meaning that one non-fixed position was set to a new random value in the range [1..9] (Table II); uniform swap mutation, where the values of two randomly selected positions of the whole grid are exchanged (Table III); and exponential probability point mutation, after each point mutation there is a 0.8 probability of a new point mutation, leading to a theoretically infinite number of mutations, but in practice only a handful (Table IV). We also tried the algorithm without any mutation at all (Table I).
  For the swap space, we used the row swap mutator, where two non-fixed positions in a row were exchanged (Table VI). We also used an exponential probability row swap mutator (Table VII), and tried the various crossover operators without any mutation (Table V).
- *Stopping criterion*:
  The stopping criterion was that no progress had been made (no fitter candidates discovered) for 20 generations.

We further tested all the mutation operators, including the smartsqaure, as local search operators. In other words, we used these operators as bases for hill-climbers. Initialization and fitness function for the hill-climbers were the same as for the population-based algorithms, and the stopping criterion was that no progress had been made for 100 000 fitness evaluations.

B. Test Environment

Five initial sudoku grids were taken from the sudoku problem generator at *www.sudoku.com*. Three of them were classified (by the generator program) as easy, one as medium hard and one as hard. We then ran 30 evolutionary runs of each combination of crossover and mutation operator on each initial grid. As time efficiency was not a main concern, we did not measure the time taken for each run.

C. Results

The result tables are organized by mutation type.

1) Hamming space: In Tables I, II, III and IV, we report the results for mutations and crossovers associated with the Hamming space. The best crossover, independent of mutation operator, is whole-row crossover, followed by two-point crossover. But even with row-wise crossover, we only relatively rarely manage to reach the optimal solution. We have never evolved an optimal solution without crossover, or when using uniform crossover.

2) Swap space: In Tables V, VI and VII, we report the results for mutations and crossovers associated with the Swap space. In general, the final fitnesses are a lot higher for the swap space experiments than for the Hamming space experiments. For the three easy instances, all the crossovers generally perform very well, when supplied with some mutation. PMX, uniform swap and multicycle crossover perform

surprisingly well even in the absence of mutation. For the "medium" difficulty problem, no combination managed to evolve the optimal solution, but for the hard problem either uniform swap or multicycle crossover seems to perform best, in one case finding the solution 15 times out of 30.

3) Hill-climbers: In table VIII, we report the results for the hill-climbers. The smartsquare mutation solved all the easy problems instantaneously, without even starting the evolutionary process. The two other mutators that performed reasonably well were uniform swap and exponential probability row swap mutation. No mutators managed to get even close to optimal fitness for the medium and hard problem instances.

TABLE I

NO MUTATION, HAMMING SPACE CROSSOVERS. EACH CELL CONTAINS THE NUMBER OF EVOLUTIONARY RUNS OUT OF 30 THAT PRODUCED THE OPTIMAL SOLUTION, AND THE MEAN FITNESS OF THE BEST CANDIDATE OF THE LAST GENERATION OF ALL 30 RUNS (THE MAX FITNESS IS 243).

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	211	0	212	0	240	1	239
Easy 2	0	212	0	212	1	239	7	241
Easy 3	0	215	0	214	4	239	1	239
Med 1	0	210	0	210	0	235	0	236
Hard 1	0	210	0	209	0	237	0	239

TABLE II

POINT MUTATION, HAMMING SPACE CROSSOVERS.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	231	0	212	4	240	1	240
Easy 2	0	231	0	212	2	239	8	241
Easy 3	0	232	0	213	5	241	9	241
Med 1	0	228	0	210	0	235	0	237
Hard 1	0	230	0	210	0	237	1	239

TABLE III

UNIFORM SWAP MUTATION, HAMMING SPACE CROSSOVERS.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	231	0	212	2	240	5	241
Easy 2	0	231	0	212	4	238	8	241
Easy 3	0	231	0	213	4	240	14	242
Med 1	0	227	0	210	0	232	0	236
Hard 1	0	229	0	210	0	236	0	238

TABLE IV

EXPONENTIONAL PROBABILITY POINT MUTATION, HAMMING SPACE CROSSOVERS.

Grid	None		Uniform		Twopoint		Whole-row	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	222	0	212	1	238	2	240
Easy 2	0	221	0	212	1	237	8	241
Easy 3	0	221	0	214	3	238	3	240
Med 1	0	220	0	210	0	231	0	235
Hard 1	0	220	0	210	0	234	0	238

VI. CONCLUSIONS

In this paper we have designed new geometric crossovers for the Sudoku puzzle that deal in a natural way with its constraints. We have demonstrated the usage of the important notion of product geometric crossover to straightforwardly derive (i) new geometric crossovers for the entire grid obtained by employing simple geometric crossovers for each row and (ii) the distance functions associated with them. This has allowed us to analyze the geometric fitness landscape associated to the new geometric crossovers and tell *a priori*, by the way the fitness landscape is constructed, that the new crossovers are very well-suited to the Sudoku puzzle hence likely to perform well. We extensively tested a number of geometric crossovers, mutations and hill-climbers and found that the operators associated with the row-swap distance are the best and produce consistently (near) optimal Sudoku grids. In future work, we want to extend our analysis to Sudoku grid of increasing size and combine smartsquare mutation with the best geometric crossover to tackle bigger and tougher instances.

REFERENCES

[1] M. Deza and M. Laurent. *Geometry of cuts and metrics*. Springer, 1991.

[2] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[3] T. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, University of New Mexico, 1995.

[4] D. E. Knuth. Dancing links. *Preprint P159, Stanford University*, 2000.

[5] A. Moraglio. Product geometric crossover. *Technical Report CSM-446, University of Essex*, 2006.

[6] A. Moraglio and R. Poli. Topological interpretation of crossover. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1377–1388, 2004.

[7] A. Moraglio and R. Poli. Geometric crossover for the permutation representation. *Technical Report CSM-429, University of Essex*, 2005.

[8] A. Moraglio and R. Poli. Geometric landscape of homologous crossover for syntactic trees. In *Proceedings of CEC 2005*, pages 427–434, 2005.

[9] A. Moraglio and R. Poli. Topological crossover for the permutation representation. In *GECCO 2005 Workshop on Theory of Representations*, 2005.

[10] A. Moraglio, R. Poli, and R. Seehuus. Geometric crossover for biological sequences. In *Proceedings EuroGP (to appear)*, 2006.

[11] P. M. Pardalos and M. G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.

[12] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *Preprint, University of Tokyo*, 2005.

TABLE V

NO MUTATION, SWAP SPACE CROSSOVERS.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	212	0	239	26	243	28	243	0	237	24	243
Easy 2	0	212	2	241	21	242	21	242	0	234	22	242
Easy 3	0	214	1	240	29	243	30	243	0	237	30	243
Med 1	0	210	0	236	0	237	0	237	0	230	0	237
Hard 1	0	210	0	238	9	242	15	242	0	234	12	242

TABLE VI

ROW SWAP MUTATION, SWAP SPACE CROSSOVERS.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	5	241	15	242	28	243	29	243	22	242	27	243
Easy 2	3	240	16	242	24	243	27	243	9	241	23	243
Easy 3	8	241	26	243	30	243	30	243	25	243	30	243
Med 1	0	236	0	237	0	237	0	235	0	237	0	237
Hard 1	0	239	5	241	4	241	9	239	1	240	10	242

TABLE VII

EXPONENTIAL PROBABILITY ROW SWAP MUTATION, SWAP SPACE CROSSOVERS.

Grid	None		Whole-row		PMX		Uniform swap		Onecycle		Multicycle	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	239	14	242	27	243	24	242	20	242	29	243
Easy 2	2	239	16	242	19	242	21	241	11	242	23	243
Easy 3	5	240	27	243	29	243	30	243	29	243	29	243
Med 1	0	234	0	237	0	234	0	230	0	236	0	237
Hard 1	0	237	2	241	0	236	1	234	2	239	7	241

TABLE VIII

HILL-CLIMBERS. MUTATION OPERATORS ALONG THE Y-AXIS.

Grid	None		Point		Uniform swap		Exp. p. point		Row swap		Exp. p. row swap		Smartsquare	
	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg	#opt	avg
Easy 1	0	190	0	232	12	241	0	233	1	238	7	241	30	243
Easy 2	0	190	0	232	9	241	0	235	0	237	2	239	30	243
Easy 3	0	188	0	232	21	242	0	233	6	239	15	241	30	243
Med 1	0	182	0	230	0	236	0	232	0	234	0	236	0	235
Hard 1	0	171	0	232	1	240	0	233	0	238	0	239	0	239