LandscapeEC: Adding Geographical Structure to Cellular Evolutionary Algorithms

Lucas Ellgren and Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN 562367
ellgr001@morris.umn.edu
mcphee@morris.umn.edu

Abstract

Evolutionary computation is a field of Computer Science in which possible solutions to a problem are represented as individuals and undergo the basic principles of recombination and mutation to evolve an optimal or near-optimal solution to a problem. In this paper we introduce LandscapeEC, a new type of evolutionary algorithm (EA) that introduces the concept of geography to more effectively evolve solutions to problems.

In cellular EAs, individuals are placed in cells on a grid, and can only reproduce with nearby individuals. LandscapeEC is a type of cellular EA which uses geography to add a landscape to the grid. In real-world biology, organisms in different locations must adapt to different conditions (e.g., higher temperatures, less rainfall) in order to survive. In LandscapeEC, geography introduces restrictions that individuals must meet in order to live in each cell. These restrictions are usually sub-problems derived from the chosen problem. By arranging these restrictions in the landscape, we can encourage individuals to gradually make their way across various sub-problems of roughly increasing difficulty of survival, drawing closer to a full solution as they do.

Here we present the results of experiments with several different kinds of geography, including flat geography, gradient geography and fractal geography, all applied to square 2-D grids. Flat geography is essentially the same as traditional cellular EAs; individuals do not need to meet any restrictions to live. Gradient geography creates a smooth gradient of cells that gradually increase in difficulty as they move closer and closer to the center of the grid. Fractal geography is similar to gradient geography, but the gradient is generated using fractals with random noise to create a rough, unsymmetrical landscape.

In this initial study we used 3-SAT as our test problem, as it is an NP-complete optimization problem which can be easily broken down into sub-problems. Our comparison of these different kinds of geography will determine if adding geography to cellular EAs will increase their effectiveness, efficiency and ability to maintain a diverse population.

1 Introduction

Evolutionary Computation (EC) is a branch of artificial intelligence that utilizes the basic principles of biological evolution in order to find optimal or near-optimal solutions to problems. In regular biological systems, organisms become better suited to their environment through a gradual process of changes. Such changes usually occur as genes are passed down from parents to children with minor mutations. If the resulting organism is better suited to its environment, then it has a greater chance to live on and spread its genes to the next generation. In EC systems, we represent potential solutions as individuals, and rank them based on their *fitness*. Here, fitness refers to how well the individuals solve the chosen problem. Individuals which have a higher fitness obtain a greater chance of reproducing and spreading their genes to the rest of the population. Individuals which have lower fitness may not be able to reproduce, and their potential solutions die off. Through this Darwinian-like process of survival of the fittest, we hope the system will create high-quality solutions to the problem at hand.

Reproduction and the passing of genes down through generations is the main engine that drives Evolutionary Algorithms (EAs). Reproduction can occur in a variety of different ways, but generally it follows a straight-forward pattern. First, the system determines how many new individuals must be created for the next generation. Many EAs put a maximum on the number of individuals that can be created to prevent the population from growing too much. After the system knows how many individuals to create, it begins selecting individuals to be the parents for a new individual. How individuals are selected can be random, or it can be based on the fitness of the individuals. Then parts are taken from the genes of the parents and combined to make a new set of genes. This operation is called a crossover operation. Finally, the genes of the new individuals undergo random mutation. Mutation is a small change in one or more parts of the genes; it is crucial for maintaining diversity in the population and exploring new potential solutions. After all of this is done, the new individual is placed in the population and the process repeats again.

Cellular Evolutionary Algorithms (cEAs) are EAs that divide their population into cells which are then arranged in a variety of ways, like 2D grids, 3D grids or undirected graphs. Cells in a cEA can contain either one or many individuals depending on the implementation. The maximum number of individuals a cell can contain is called its *carrying capacity*. In cEAs, individuals can only interact with other individuals in their cell or neighboring cells. Individuals in cEAs interact with each other similar to most EAs, however mate selection and reproduction can only occur in the neighborhood of an individual's cell.

For cEAs to work effectively, individuals must be able to move between cells. This can happen in one of two ways. The first option is to have all children be born in a random location within the neighborhood of their parents after reproduction occurs. The other option is to allow individuals to migrate between cells and have their children be born in the same cell as their parents. cEAs that use the second option are said to have *migration* incorporated in their algorithm. Migration occurs randomly, with every individual given a random chance to move every generation. Usually individuals are only allowed to move to a cell directly neighboring theirs, but some cEAs allow individuals to migrate to cells 2 or more spaces away [1].

The advantage of cEAs over EAs is their ability to maintain diversity. Regular EAs have

panmictic mating, which means that any individual has the ability to mate with any other individual. This can lead to an early convergence on a non-optimal solution within the problem, since individuals with higher-than-average fitness will dominate the population and potentially useful alternate solutions get crowded out. Cellular EAs instead have only local mating, which slows the spread of slightly better solutions in the population. This allows the algorithm to explore a greater variety of possible solutions instead of rapidly converging on local optima[1].

2 Background

2.1 Basic Components of LandscapeEC

The main purpose of LandscapeEC is to model the effect of geography on cellular evolutionary algorithms, however most of its underlying structure is built upon already existing components of EC systems. The simplified structure for the runs are detailed in Algorithm 1. In this section, we will go over the basic components of LandcapeEC in detail and describe how they work together to form a functioning evolutionary algorithm.

The most important factor in any EC system is the reproduction of individuals. In LandscapeEC, we first calculate the number of individuals needed for the next generation, and then generate them using three main operations: selection, crossover and mutation. The algorithm determines how many new individuals each cell need through the Reproduction Rate parameter. This is a value usually between 1 and 5 that determines how many new individuals must be made for each existing individual in a cell (cells are explained in further detail in section 2.2).

When the algorithm needs to create a new individual, it must first *select* the individuals to use as parents. In LandscapeEC we use Tournament Selection with a tournament size of two. This operation grabs two random individuals from the neighborhood of the current cell and selects the one with the one with the better fitness to be the first parent. The process is repeated for the second parent. Next, crossover occurs, using a Uniform Crossover operation. This process generates a new bitstring by randomly pulling bits from either of

[1] A local optimum is a potential solution that has a very high fitness, but is not the optimal solution to the problem [3]. EAs tend to focus on local optima instead of exploring other potential solutions, causing them to become "stuck".

Algorithm 1 Pseudocode for the main run loop of LandscapeEC

```
runGenerations()
  Initialize Starting Population and Grid
  while Best Fitness < 1.0 AND Function Evaluations Limit Not Reached do
    Perform Migration
    Perform Draconian Reaper
    Perform Elitism
    Perform Reproduction
  end while
```

the two parents. For every position in the bitstring, a coin is flipped. If it comes up heads, then the bit from the first parent is put into the new individual at that point, if it comes up tails, then the bit from the mother is used. Lastly, mutation occurs in the new individual through Point Mutation. When this happens, each bit is given a random chance of being flipped - changing from a 0 to a 1 or vice versa. The chance a bit will flip is determined by the Average Mutations parameter, which specifies how many bits will flip on average per individual. After all three processes are finished, the individual is placed into the next generation of the population.
Another useful process that we use in LandscapeEC is Elitism. While not a vital aspect to the algorithm, this process allows us to make steady progress without losing valuable individuals. Elitism simply saves a certain percentage of individuals with high fitness levels within each cell to keep around for the next generation. How many individuals we keep is dependent on the Elite Proportion parameter in our code. Elite members are still allowed to participate in reproduction. However, all members of the non-elite will only live on to the next generation through their children.

2.2 Cellular Components of LandscapeEC

LandcapeEC, being a cellular evolutionary algorithm, also has components typical of cEAs. It currently supports 2-D worlds (grids) divided into square cells. The dimensions of the world are specified by the parameter World Dimensions. When we generate the starting population within the world, we allow them to be placed within cells according our parameters. If we set the Starting Location parameter to be 'Origin', then only the cell at coordinates (0,0) will be filled with individuals. If Starting Location is 'Corners' then each of the four corner cells of the grid will be filled with individuals, and if Starting Location is 'Everywhere' then every cell within the world will be filled with individuals. Individuals are generated with completely random bitstrings.
Without migration, individuals would never spread beyond the cells that they start in. In our algorithm, we perform migration first before any other operation in the current generation (see Algorithm 1). Every individual is given the same random chance to migrate, determined by the parameter Migration Probability. Migration Probability is represented as a value between 0 and 1, with 1.0 being a 100% chance for an individual to migrate and 0.0 being a 0% chance. Migration Distance is the parameter we include to determine how far individuals are allowed to migrate. This can be any integer greater than or equal to zero, so for example a value of 2 would allow individuals to migrate to a cell up to 2 spaces away. Individuals are also allowed to migrate to cells diagonally around them. Unlike other cEAs, newly created individuals are not placed randomly within the neighborhood of a cell, they will always be placed in the same cell as their parents.
After migration happens, a specialized operation called Draconian Reaper occurs. This operation requires Geography within the world in order to have an effect. This operation is explored in detail in Section 3.1.

2.3 The Boolean Satisfiability Problem

The Boolean Satisfiability Problem, also known as SAT, is an NP-Complete, combinatorial optimization problem [7]. A SAT problem is defined as having a set number of boolean variables and a set number of clauses which use said variables. For the problem to be solved, truth values must be assigned to the set of variables or negated variables that satisfy each clause. For example, suppose that we have SAT problem E and variables x_1, x_2, x_3, x_4. Suppose that E is defined as: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$. In this example, there are two clauses. Clauses are a set of variables, which can be negated or not, joined through the $\vee$ (OR) operator. The same variable is allowed to appear in different clauses. The first clause, $(x_1 \vee x_2 \vee x_3)$ has no negated variables. For this clause to be satisfied, only one of the variables needs to be TRUE. In the other clause, $(\neg x_1 \vee x_3 \vee \neg x_4)$, the variables x_1 and x_4 are negated, so this clause can be satisfied if either variable is FALSE or if x_3 is TRUE.

Many people working with EAs have used the SAT problem [1, 2] especially 3SAT problems. 3SAT problems are SAT problems in which clauses have exactly three variables each. Most EAs that work with SAT represent potential solutions to the problem as a string of bits. Each bit can be either 1 or 0, corresponding to a TRUE or FALSE value for the variable in the SAT problem. So for a SAT problem with 50 variables, an individual in the EA would be represented by a string of 50 bits.

For EC systems using SAT, including LandcapeEC, the fitness of an individual is represented by the number of clauses the individual satisfies, divided by the total clauses in the problem. Represented mathematically, this function can be shown as follows: $F(b) = \frac{e}{t}$ where b is an individual's bitstring, e is the number of clauses it solves and t is the total number of clauses in the SAT problem. So an individual's fitness can be any number between 0 and 1, where 0 is an individual which satisfies no clauses and an individual with a fitness of 1.0 has found an optimal solution. Note that it is possible for two individuals to have the same fitness, but satisfy different clauses.

3 Geography

3.1 The Geography Concept

Geography is the main concept behind LandscapeEC. It has been explored previously [4, 6], but it remains a relatively new concept in the field of EC. In most cEAs, individuals are allowed to live within any cell as long as they can migrate to it. However, when Geography is applied to a grid, certain cells have requirements added to them. For an individual to live in a given cell, it must meet the requirements. This parallels well known biological concepts. In the real world, organisms must adapt to their environments in order to live in them successfully. For example, a creature suited to a tropical rain forest would have to change many of its biological features in order to live and thrive within a desert.

How restrictions are applied to cells within the grid of a cEA can vary, but for the SAT problem used in this study it is relatively simple. A 3SAT problem can easily be broken into sub-problems by only checking for a particular subset of clauses. When we put a geographical restriction on a cell, we require that individuals satisfy a certain group of clauses

in order to live within the cell (see Section 2.3 for more information on satisfiability). However, checking to see whether or not an individual meets the requirements of a cell does not affect its overall fitness. An individual can have a very high fitness but still be unable to meet the requirements of the cell its in.
The way we enforce the requirement of the cells is through the Draconian Reaper operation. This operation is aptly named for its zero-tolerance way of dealing with individuals that are not up to par with a cell's requirements. When an individual has migrated to a cell and does not satisfy *all* of the clauses it requires, the Draconian Reaper simply removes it from the population altogether. If an individual doesn't meet the requirements of a cell it's just migrated to, it will be reaped and will not get a chance to reproduce within the cell.
The hypothesis is that adding Geography to a cEA will allow it to maintain a higher level of diversity, and to find the solutions to difficult clauses quicker than most other cEAs. By adding selection pressure for only certain clauses, we force individuals to solve them in order to spread to other areas in the grid. Also, by initially separating populations and letting them slowly merge together, interesting combinations between populations that solve different clauses may occur.

3.2 Types of Geography

There are a multitude of ways to construct geography within a 2D grid, and during the production of LandscapeEC we explored several different types of geography, eventually focusing on the two most interesting that we had discovered. We wanted to focus on creating geography that encouraged individuals to solve difficult clauses while not restricting their size too much. This meant that the transition from easy cells to hard cells had to be gradual. We call cells 'easy' when they do not have many clauses required to live in them, and 'hard' when they require a lot of clauses. The two types of geography that we focus on in this paper are Gradient Geography and Fractal Geography.
Gradient Geography is named for the smooth transition from easy to hard cells it creates. It features four corner cells which have no clause requirements, and a cell in the center in which all clauses are required. Between the corners and the center there is a smooth gradient of cells that go from 'easy' to 'hard' (see Figure 1). In order to construct this geography, we first make a list of all the clauses for the given 3SAT problem. Then, for each cell, we compute its distance to the center square. This distance is divided by the maximum distance to get a percentage, which represents what percentage of the clause list will be required for a given cell. For example, if we have a clause list consisting of clauses c_1, c_2, c_3 and c_4, then a cell which is 50% of the way from the corner and the center will have c_1 and c_2 as required clauses.
Fractal Geography is very similar to Gradient Geography in that it has a gradient of cells going from easy in the corners to hard in the center. However, Fractal Geography is created recursively, by repeatedly subdividing the grid into subsections and adding random noise. This is similar to how fractal terrains are created in the world of 3D modeling [5], and gives the grid a rough texture (see Figure 2). Fractal geography is constructed by first dividing the grid into four quadrants. In each quadrant, the cell on the outer corner of the world is given no clause requirements and the cell in the center of the world is given total clause requirements. The other remaining corners of the quadrants are given half the clauses of

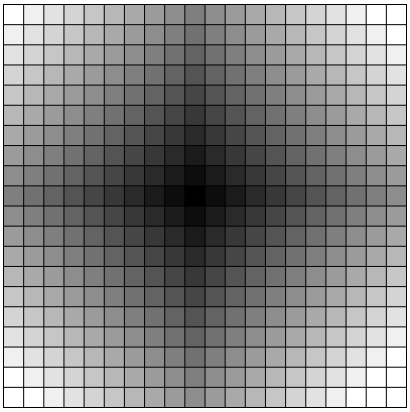

Figure 1: An example of Gradient Geography applied to a 20×20 grid. The darkness value of each cell represents how many clauses it requires.

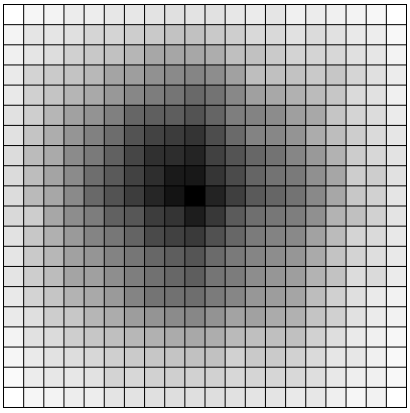

Figure 2: An example of Fractal Geography applied to a 20×20 grid, with a random noise factor of 5. Notice its rough, unsymmetrical gradient. The darkness value of each cell represents how many clauses it requires.

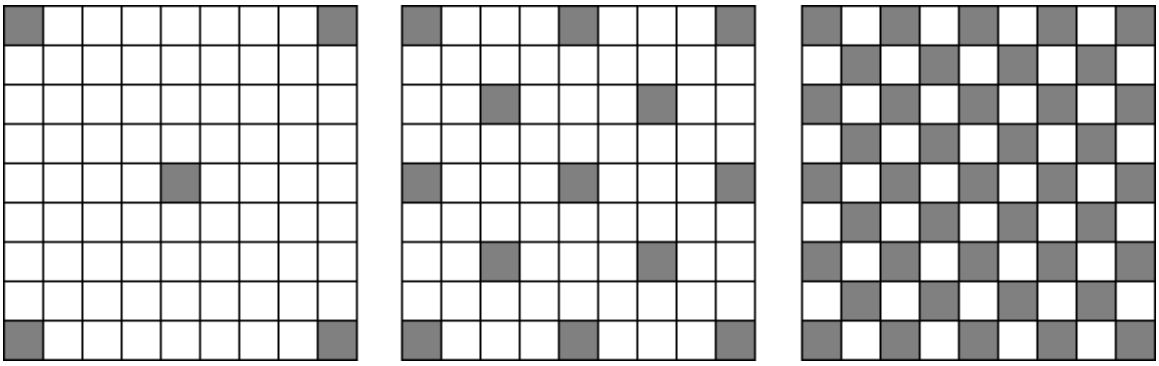

Figure 3: A visual representation of the order in which clause requirements are assigned to cells in a grid for the generation of Fractal Geography. Grey cells represent those which have been assigned requirements.

the maximum, with a few clauses randomly added or removed to create noise. Then each quadrant is divided into sub-quadrants, and the corner cells within are given half the clause requirements of the larger quadrant (plus or minus a few clauses). This process of subdivision and assigning clause requirements repeats until all cells in the world have been filled. See Figure 3 for a visual representation of this process.

4 Experimental Setup

In this section, we go into detail about the set of parameters we used for our experimental runs. For our runs, we focused on comparing four different versions of LandscapeEC: Non-Cellular, No Geography, Gradient Geography and Fractal Geography. *Non-cellular* LandscapeEC mimics a simple non-cellular EA, to use as a point of comparison. Its grid has only 1 cell, and no migration or reaping occurs. *No Geography* simulates a common cEA without any geographical features applied. Migration occurs in this set-up, but no reaping of individuals happens. *Gradient Geography* and *Fractal Geography* utilize the geographical features of LandscapeEC. Both migration and reaping occur in these setups.
We tested each version on five different 3SAT problems. For each combination of version and problem, we performed 100 runs. Runs end when they exceed the limit on function evaluations, or when an optimal solution is found. The five different 3SAT problems, and their relative difficulty, are listed in Table 1. All of these problems were randomly generated; their difficulty is based merely on how well our algorithm performed on them, and is not an official standard. Each version of LandscapeEC used the same operators for selection, crossover and mutation, which are explained further in Section 2.1.
We wanted each version of LandscapeEC to perform at its best, so we attempted to select parameters that suited each version. The settings for some parameters clearly give an advantage to one version, but not others. For example, having a Reproduction Rate of 1.0 benefits Non-Cellular and No Geography versions, but weakens the effectiveness of the Fractal and Gradient Geography versions. This is because versions with Geography must accommodate for individuals lost through reaping by producing them at a higher rate. The parameters used for each version are specified in Tables 2, 3 and 4.

File Name	**# Variables**	**# Clauses**	**Difficulty**
uf50-0456.cnf	50	218	Easy
uf75-015.cnf	75	325	Medium
uf75-05.cnf	75	325	Medium-Hard
uf75-090.cnf	75	325	Hard
uf100-0193.cnf	100	430	Medium

Table 1: The set of 3SAT problems used. Difficulty is relative to how well our algorithm performs on them.

Parameter	Value
Average Mutations	1
Reproduction Rate	1.0
Carrying Capacity	4000
Elite Proportion	0.1
World Dimensions	*N/A*
Toroidal	*N/A*
Starting Population	*N/A*
Migration Probability	*N/A*
Migration Distance	*N/A*

Table 2: Parameters used for non-cellular LandscapeEC.

Parameter	Value
Average Mutations	2
Reproduction Rate	1.0
Carrying Capacity	6
Elite Proportion	0.1
World Dimensions	20×20
Toroidal	False
Starting Population	Corners
Migration Probability	0.1
Migration Distance	1

Table 3: Parameters used for cellular LandscapeEC with no geography.

Parameter	Value
Average Mutations	1
Reproduction Rate	3.0
Carrying Capacity	10
Elite Proportion	0.2
World Dimensions	20×20
Toroidal	False
Starting Population	Corners
Migration Probability	0.1
Migration Distance	1

Table 4: Parameters used for cellular LandscapeEC with gradient geography and fractal geography.

5 Results

In Table 5, we compare the success rates for each version of LandscapeEC. If an EA has a higher success rate, then it is usually implied that the EA is more effective at finding solutions. As shown in the table, No Geography had the highest success rate on average for all problems, followed by Gradient Geography, then Fractal Geography and finally Non-Cellular. Non-cellular was the only method to fail on a problem for all 100 runs, and No Geography was the only method to succeed on a problem for all 100 runs. See Figure 4 for a box-and-whisker plot of the success rates for each version.

We also compare the number of function evaluations used by each version of LandscapeEC, as seen in Table 6. We only take into account successful runs for these numbers, since unsuccessful runs always used 10 million evaluations. EAs that use a lower number of function evaluations tend to find solutions faster, and therefore it is implied that they are more efficient. As shown in the table, Gradient Geography tends to use the most function evaluations, followed by Fractal Geography, followed by No Geography and finally Non Cellular. See Figure 5 for a box-and-whisker plot of the evaluations used for each version.

Problem File	**Non-Cellular**	**No Geography**	**Gradient**	**Fractal**
uf50-0456.cnf	1%	100%	83%	92%
uf75-015.cnf	0%	73%	60%	44%
uf75-05.cnf	0%	45%	40%	20%
uf75-090.cnf	0%	20%	22%	13%
uf100-0193.cnf	55%	76%	69%	58%
Average Overall	19.6%	62.8%	54.8%	45.4%

Table 5: Success Rate comparison for the different versions

Problem File	**Non-Cellular**	**No Geography**	**Gradient**	**Fractal**
uf50-0456.cnf	4,141,000	126,600	68,200	127,200
uf75-015.cnf	*N/A*	792,400	978,700	1,632,000
uf75-05.cnf	*N/A*	716,900	1,187,000	1,724,000
uf75-090.cnf	*N/A*	450,900	538,500	675,500
uf100-0193.cnf	220,100	755,900	1,933,000	1,646,000
Median Overall	442,100	499,700	718,400	618,600

Table 6: A comparison of median function evaluations used. Only runs that resulted in successes were used in these statistics.

6 Conclusions

From our results, we see that No Geography was more successful overall at finding an optimal solution than both types of Geography. This implies that Geography, when applied to cEAs does not make them more effective. However, looking at the the number of

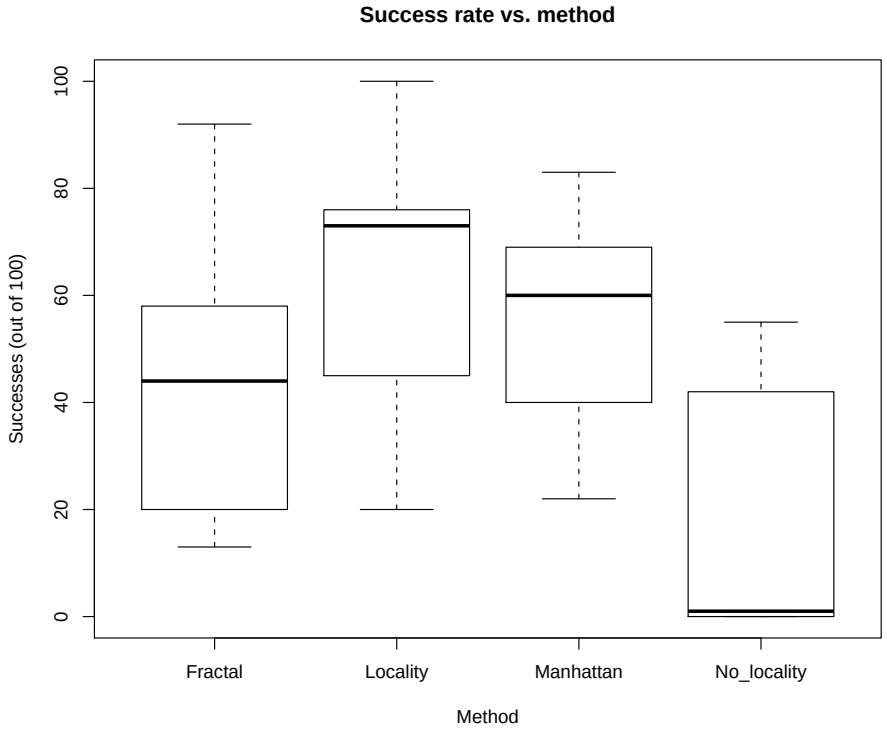

Figure 4: A box-and-whisker plot of success rates for each version.

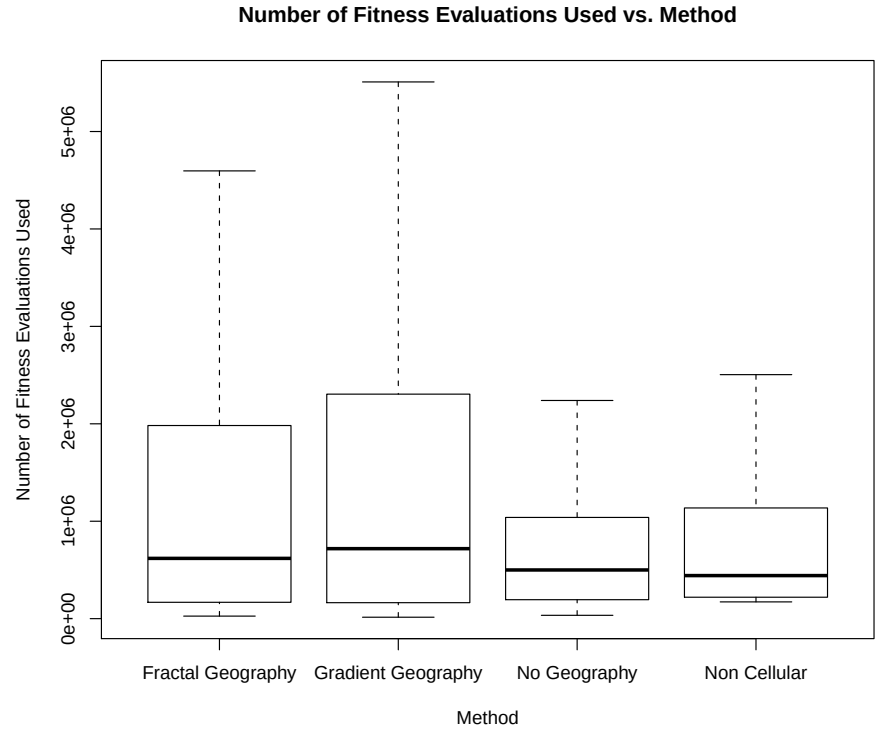

Figure 5: A box-and-whisker plot of function evaluations used for each version. Only runs that resulted in successes were used in these statistics.

evaluations used by both types of Geography revels an interesting trend. Both Gradient Geography and Fractal Geography tend to use use more evaluations when they succeed, which means that they tend to find solutions later in the runs. When No Geography and Non-Cellular succeed, they tend to do so fairly quickly, using less function evaluations. Gradient Geography in particular had one quarter of its runs finish later than the runs for No Geography (see Figure 5). Usually, EAs tend to either find the solution fairly early in a run or not at all; only rarely do they find solutions late in the run. This implies that the runs with geography had a greater amount of diversity in their populations, since a high level of diversity increases the chance of finding solutions later than average.

There is always a margin of error when dealing with EAs, and these experiments are no exception. The parameters we used for the different versions of LandscapeEC may not have been the most optimal for our runs. The problems we selected for our runs may have been too easy to adequately test the performance of cEAs with geography. There is also the chance that the runs were a fluke and that No Geography doesn't actually have a higher rate of success than Gradient or Fractal Geography. However, the results do show evidence to support the theory that Geography is able to maintain diversity better than No Geography. Additional work using Geography on problems other than SAT may potentially yield more interesting results. There is also the chance that cEAs with geography will perform more effectively on problems where most regular cEAs fail. We hope to further explore the effects of geography in future research.

References

[1] ALBA, E., ALFONSO, H., AND DORRONSORO, B. Advanced models of cellular genetic algorithms evaluated on sat. In *Proceedings of the 2005 conference on Genetic and evolutionary computation* (New York, NY, USA, 2005), GECCO '05, ACM, pp. 1123–1130.

[2] GOTTLIEB, J., MARCHIORI, E., AND ROSSI, C. Evolutionary algorithms for the satisfiability problem. *Evol. Comput. 10* (March 2002), 35–50.

[3] JOHNSON, D. S., PAPADIMTRIOU, C. H., AND YANNAKAKIS, M. How easy is local search? *J. Comput. Syst. Sci. 37*, 1 (Aug. 1988), 79–100.

[4] KORTH, A., HUTCHINSON, T., AND MCPHEE, N. On the impact of geography and local mating in evolutionary computation. In *Proceedings of Midwest Instruction and Computing Symposium* (2007), MICS '07, pp. 1–14.

[5] MARTZ, P. Generating random fractal terrain, 1996.

[6] SPECTOR, L., AND KLEIN, J. Trivial geography in genetic programming. In *Genetic Programming Theory and Practice III*, T. Yu, R. L. Riolo, and B. Worzel, Eds., vol. 9 of *Genetic Programming*. Springer, Ann Arbor, 12-14 May 2005, ch. 8, pp. 109–123.

[7] TOVEY, C. A. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics 8*, 1 (1984), 85 – 89.