LandscapeEC: Comparing 2D to 3D Cellular Evolutionary Algorithms

Nicholas Cornhill and Nicholas Freitag McPhee
Division of Science and Mathematics
University of Minnesota, Morris
Morris, MN 562367
cornh044@morris.umn.edu
mcphee@morris.umn.edu

Abstract

Evolutionary Computation (EC) is a field of Computer Science that utilizes the basic principles of biological evolution to create a system that can be used to find an acceptable solutions to problems, which are called individuals. A specific kind of EC system, called a cellular Evolutionary Computation (cEA), models space and distance to limit interaction interaction within the system in an attempt to maintain diversity in the population and increase the chances of discovering an optimal solution. In this paper, we compare two kinds of cEA systems, one that models space in two dimensions, and one that models space in three dimensions. We compare the performance of 2D and 3D worlds on several 3SAT problems and the ONESMAX problem, and find that at least in our experiments 2D worlds tended to outperform 3D worlds and that the optimal parameter settings for 3D worlds were sometimes significantly different than those for 2D worlds.

1 Introduction

Evolutionary Computation (EC) is a field of Computer Science that utilizes the basic principles of biological evolution to create a system that can be used to find acceptable solutions to problems. In biological systems, organisms better suited for their environment generally yield more descendants than those animals that aren't as suited to their environment. Over time this leads to successive populations being generally more suited for the present environment than the earlier generations. In EC, instead of organisms, we have potential solutions to a problem, which we call *individuals*. These individuals are rated on how successfully they solve the given problem, or how close they get to solving the problem. This rating is called their *fitness*. Successive generations are simulated one after another, which would be similar to time passing in a biological system. The more fit individuals are

selected to generate the next generation of individuals by using various systems that mimic sexual reproduction, genetic mutation, and the theory of the survival of the fittest.

EC systems can be used to solve a variety of practical problems and have the added benefit of not requiring much human effort since after the initial set up all of the computation is done by a computer. The only restrictions on what kind of problem an EC system can solve are that a computer has to be able to generate potential solutions as well as evaluate the fitness of potential solutions. EC systems generally do well solving symbolic regression problems, combinatorial optimization problems, such as SAT [3], as well as more practical problems such as soft sensors in chemical plants, and radio antennae for NASA [5].

Cellular Evolutionary Algorithms (cEAs) are EC systems that separate the population into cells which are connected in certain ways, like 2D grids, 3D grids, or other graphs [1]. These cells are also called *locations*. Individuals in a cell interact only with individuals in their own cell or individuals in their neighborhood. The primary method in which cross-cell interaction occurs is by migration in which an individual moves from one cell to another neighboring cell. The key advantage of implementing a cEA instead of a regular EA is to help maintain diversity, or the number of different kinds of solutions that are present in the system [1]. Having a diverse population has a number of beneficial effects on the system that often lead to acceptable solutions being found more consistently as well as faster.

This paper will explore the differences between cEA systems that are based on both 2D and 3D cuboids. A description of how the populations are first initialized, as well as how subsequent populations are generated will be given in Section 2.1, followed by a more in depth description of 2D and 3D worlds in Section 2.2, and a description of the problems that are used in the experiments in Section 3. Then we will present the results of our experiments in Section 5 and a discussion of those results, as well as other possible research topics suggested by our results will be in Section 6.

2 LandscapeEC

In this section we will describe the LandscapeEC [4] [2], system, which we used to explore the impact of different shapes of worlds. The system also supports a variety of parameters that can be changed to adjust the behavior of the system in various ways. How the world is set up differs depending on the shape of world, or whether or not the world is 3D or 2D. But once the world is formed, the processing of generations, as described in Algorithm 1 is the same. This section will be devoted to exploring the details of how this LandscapeEC system works. First, how the initial and subsequent populations are generated will be described in section 2.1. Then, the specifics of 2D and 3D worlds will be presented in section 2.2.

Algorithm 1 Pseudocode for the main run loop of LandscapeEC

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

2.1 Running a Generation

One of the major aspects of an EA system is how it deals with generating a population. The first step in our EA system is the generation of the initial population. In LandscapeEC our individuals are always bitstrings of a length determined by the problem we are addressing. A number of cells are selected, determined by the *starting locations* parameter, and those locations are filled to capacity with a number of individuals, the capacity being determined by the *carrying capacity* parameter. These individuals are generated randomly; each bit of each individual is selected uniformly from the set 0,1.

After the initial population of the world, Algorithm 1 is run until one of the two termination parameters is met. One of the termination conditions is that an individual has been found that can solve the problem successfully. The other one is that the number of evaluations has reached its limit, as set by the *maximum number of evaluations* parameter [3].

The first step of Algorithm 1 is to migrate individuals. Each location has a list of locations that it is connected to called its *neighborhood*. The probability of an individual migrating is determined by the *migration probability* parameter, which is a value from 0 to 1, which determines the probability of an individual migrating, with 0 meaning that no migration occurs, and 1 meaning that an individual will always migrate. If the individual has been selected to migrate, a location in its neighborhood is randomly selected, and the individual will be moved to that location for the next generation.

The next step in the system is performing *elitism*. Elitism copies a certain number of the best individuals at any given location over to the next generation. This is to stop key individuals from dying out or the system from moving backwards. The number of individuals to copy is determined by the *elite proportion* parameter.

The final step is reproduction, or the generation of new individuals. The first step of this is to determine how many new individuals need to be generated. The parameter *reproduction rate* determines the maximum number of individuals that can be generated. The number of individuals in the cell is multiplied by the reproduction rate, that value is the number of individuals that will be generated unless that number would cause the location to have more individuals than the carrying capacity. Then, only enough individuals to fill a location are generated.

Once the number of individuals to be generated is determined, the actual production of indi-

Algorithm 2 Pseudocode finding the neighborhood list in a 3D world. x, y, and z are the 3D coordinates of the location in the world.

```
get3DNeighborhood(x, y, z
  for max(x − 1, 0) ≤ i ≤ min(x + 1, width − 1) do
    for max(y − 1, 0) ≤ j ≤ min(y + 1, depth − 1) do
      for max(z − 1, 0) ≤ k ≤ min(z + 1, height − 1) do
        addToNeighbors(i, j, k)
      end for
    end for
  end for
```

viduals is done. The first process in generating an individual is called tournament selection. We select two pairs of two individuals randomly, and the better of each pair is selected to be a newly generated individual's parent. Once the parents are selected, crossover occurs using the two parents. Each bit of the new individual is chosen by selecting one of its parents randomly and using the bit they have in the given position for their children. This process is repeated for every bit the individual has.

Once crossover is finished, the individual is mutated. Each bit of the individual has a chance to be flipped. This chance is determined by the *average mutations* parameter, divided by the number of bits in an individual. Therefore, on average, an individual will have as many bits flipped as the value of the average mutations parameter. Once mutation occurs, the individual is evaluated for its fitness against the problem that is trying to be solved. If the individual solves the problem, the run ends in success. If the number of individuals evaluated reaches the value of the *maximum number of evaluations* parameter without finding a solution, the run ends in failure.

2.2 Shaping the World

The second major part of our LandscapeEC system is the shape of the worlds and how they are set up. In these experiments, there are two kinds of worlds: 2-dimensional (2D) and 3-dimensional (3D) worlds. 2D worlds are described by giving its size along its two dimensions. Then, the world is generated from those dimensions, shaped like a lattice where each intersection is a location. The neighborhood of a location are the immediately surrounding locations, i.e. the points next to a given point and the points diagonally adjacent to the point, see Figure 1. Therefore, the maximum number of neighbors a given location can have would be eight, although locations on the edge of the world would have less. In some systems neighborhoods can be expanded to include more distant cells, but that option isn't explored in this paper.

A 3D world is defined by the size along its three dimensions. Each location has a number of neighbors, which are all of the locations immediately surrounding it as illustrated in Algorithm 2. Therefore, a location can have up to 26 neighboring locations, but locations on the edge of the world may have fewer.

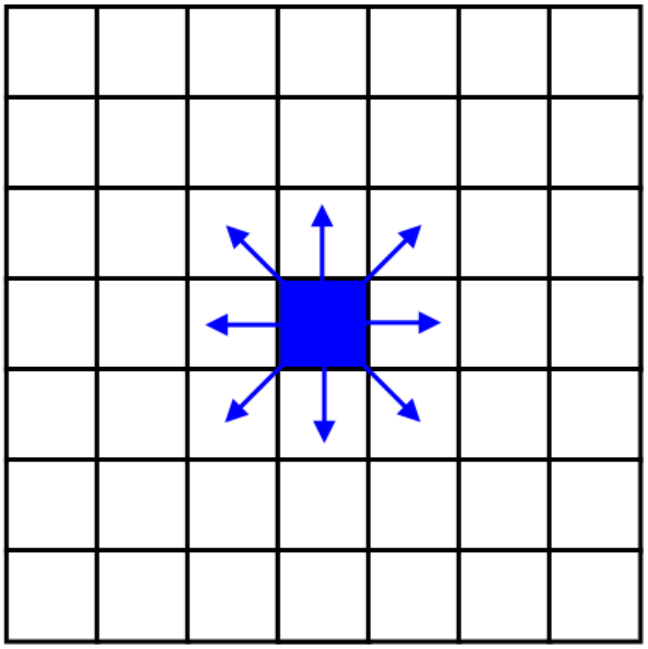

Figure 1: An illustration of a 2D Neighborhood. Note, if a location is on the edge of the world, it will have less than eight neighbors.

In addition to neighborhoods, another difference between 2D and 3D worlds is starting locations. The *starting location* is a parameter that determines where individuals are initially generated. The two options are origins and corners. The origin in a 2D world is the upper left corner, while in a 3D world, it is the upper left corner of the top layer of the world. The second option for starting location is corners. In 2D world, the corners are the four corners of the lattice, while in a 3D world it is the eight corners of the cuboid.

3 Test Problems

The experiment described here use two problems: 3SAT and ONESMAX. Solutions to these problems are naturally expressed as bitstrings, and they span a broad range of difficulties. ONESMAX is trivial to solve, and 3SAT problems range from easy to extraordinarily difficult. The specifics of their properties, as well as their implementation in our system are described below.

3.1 The 3SAT Boolean Satisfiability Problem

The Boolean Satisfiability Problem, or SAT is an NP-Complete problem where the goal is to find an assignment of values to boolean variables that satisfy each clause in a set of clauses. In 3SAT each clause consists of 3 terms each of which is a variable or a negated variable. A clause is said to be satisfied if at least on of its terms is true. A potential solution consists of an assignment of either true or false to each of the variables. A solution that solves the problem, will be one that satisfies every clause.

For example, consider SAT problem E, which has four variables, x_1, x_2, x_3, x_4 and the the two conjoined clauses $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee \neg x_4)$. For this problem to be solved, both clauses must be satisfied. For a clause to be satisfied, at least one of its terms must be true. So, for example, say we have a potential solution A, which consists of the variable settings $x_1 =$ true, $x_2 =$ false, $x_3 =$ false, and $x_4 =$ false. The first clause would be satisfied because it contains x_1, which is true. The second clause would also be satisfied, because x_4 is false, but the variable in the clause is negated, so $\neg x_4$ returns true, which satisfies the clause. As both clauses are satisfied, this assignment solves the problem.

Here individuals are represented by bitstrings whose length is equal to the number of variables in the SAT problem [3]. Each bit represents whether or not a given variable is assigned true or false. The fitness of a potential solution is determined by dividing the number of clauses satisfied by the number of total clauses. Therefore, fitness is always in the range [0,1], representing the proportion of clauses solved. It is important to note that two individuals may have the same fitness but satisfy different clauses.

In this paper we use three different 3SAT problems as described in Section 4.

3.2 ONESMAX Problem

The ONESMAX problem is a simple problem where the goal is to maximize the number of ones in a bitstring [1]. Different ONESMAX problems consist of different lengths of bitstrings. The fitness is simply the number of 1s in the bitstring divided by the length of the bitstring. In this paper, the only ONESMAX problem is one of size 100.

4 Experimental Setup

This section will explain the specifics of the various configurations that were used for this research. For our runs, we used two different kinds of worlds, 2D worlds and 3D worlds. We also had two kinds of problems, the SAT problem, and the ONESMAX problem. This means that there were essentially four main groups of runs, as both kinds of worlds were run with both kinds of problems.

We tested three different kinds of SAT problems, see Table 1, and the length 100 ONESMAX problem. The SAT problems were selected from various databases and the difficulty

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

Table 1: The 3SAT problems we ran. The difficulties are as reported in [2].

2D World Size	# of locations	3D World Size	# of locations
10 by 10	100	4 by 5 by 5	100
15 by 15	225	6 by 6 by 6	216
24 by 24	576	8 by 8 by 9	576
27 by 27	729	9 by 9 by 9	729

Table 2: The various world sizes we tested.

assigned to them is taken from [2].

We selected three different world sizes for both the 2D and 3D worlds. We used world sizes that were squares or near cubes while also picking sizes that had similar numbers of total locations. The world sizes, as well as how many locations each had are described in Table 2.

Sometimes a specific set of parameters works very well for a certain problem and world type, but works poorly for another. So we explored a variety of parameters for both worlds in an attempt to reduce parameter bias. We selected a number of parameters to vary from run to run so that we could compare "good setups" to other "good setups". The parameters we chose to vary and how we varied them are explained in the Table 3 and were chosen based on our experience with previous experiments with LandscapeEC.

There were 512 different problem and parameter settings. Each run with a specific set of parameters was run 30 times which means there was a total of 15360 runs.

Parameters	
Average Mutations	1, 2
Carrying Capacity	15, 30
Reproduction Rate	1, 3
Starting Population	Origin, Corners

Table 3: On the left are the various parameters we varied and to their right are the different values we assigned them.

	3SAT		ONESMAX	
	2D	**3D**	**2D**	**3D**
World Size	24x24,27x27	8x8x9, 9x9x9	*	4x4x5
Starting Locations	Corners	Corners	Origin	*
Carrying Capacity	*	*	*	15
Reproduction Rate	*	*	1	*
Average Mutations	*	*	1	*
Success Rate	69%	* 64%	100%	100%
Median # of Evals	3,958,000	4,709,000	49,890	42,950

Table 4: This table diagrams the best configurations and their success rates and median number of evaluations.

5 Results

Table 3 lists the most successful parameter values of those we explored , as well as the success rates and median evaluations used for those parameters. Asterisks indicate parameter settings whose values didn't make a statistically significant difference for that configuration. Figure 2 shows the distribution of completed evaluations for the best parameter values.

On the 3SAT problems 2D worlds did significantly better than 3D worlds; they succeeded more times and faster. A Kruskal-Wallis test confirms that the difference is significant with a p-value of 2.327e-8. Interestingly for both 2D worlds and 3D worlds large world sizes and starting in the corners were important parameters, while the other three parameters were negligible in effect.

In an unexpected twist, 3D worlds did solved the ONESMAX problem faster than 2D worlds, although they both had the same success rate of 100%. Also, where before the important parameters were the same when solving 3SAT. The difference in speed was confirmed to be statistically significant with a p-value of 2.2e-16. 2D and 3D worlds have completely different parameters that are important in ONESMAX. 2D worlds found reproduction rate, starting locations, and average mutations to be important, while 3D worlds only found world size and carrying capacity to be important.

6 Conclusions

We suspect that the reason why 2D worlds do better than 3D worlds when solving 3SAT is a matter of distance and diversity. 3SAT problems tend to to have a large number of local optima that EC systems can fall into. Because of this, maintaining diversity is key. This may suggest why corners are so important, there needs to be more than one starting population because there's a strong chance that each population is very different from another. However, diversity isn't maintained in 3D systems due to the distance between starting

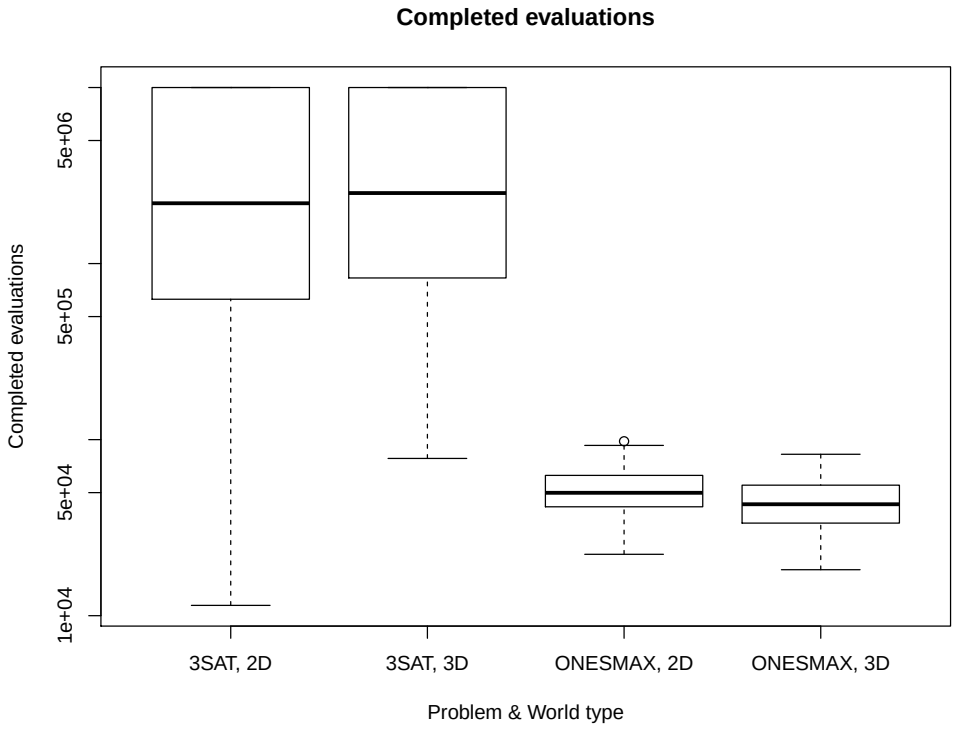

Figure 2: A box-and-whisker plot of the number of completed evaluations for each version. Note the log-scale on the y-axis.

locations. In the largest 3D world, the distance between a starting location and another starting location was as small as nine locations. Meanwhile, in a 2D world, the distance between starting locations was a factor of three larger, with the the largest world being 27 across. What is happening is a local optima is found, which can spread much more quickly in a 3D world than in a 2D world due to the smaller distances between any two locations in the 3D worlds. This compromises diversity and caused 2D worlds to perform better in our runs.

Meanwhile, the ONESMAX problems are completely different. They have no local optima to fall for, so the system has a different approach to solving them. In 3D worlds, smaller potential populations are better because you want the number of new individuals to be minimized, to a point. The problem will often be solved so quickly that new individuals don't have time to "mature" and contribute to solving the problem.

In 2D worlds, world size doesn't matter as much, because the world almost never fills before the problem is solved. Origin is important because the problem is solved so fast that corners never have time to spread out fast enough to interact meaningfully, so starting in one spot saves evaluations.

Overall, although 3D worlds didn't solve 3SAT faster, it illustrated the importance of distance in cEA systems. Having a large distance between starting locations is good for maintaining diversity, while if maintaining diversity isn't an issue, then more starting locations

are better. A potential route of future research is to decrease the number of connections between locations and try to spread them more thinly. This is essentially trying to do the opposite of what 3D worlds do, where there is an increased number of connections as well as shorter distances between locations.

Acknowledgements

Many thanks to Lucas Ellgren for his feedback and for the use of several diagrams.

This work was supported in part by the Morris Academic Partners program at the University of Minnesota, Morris.

References

[1] ALBA, E., AND DORRONSORO, B. *Cellular Genetic Algorithms*. Springer, 2008.

[2] ELLGREN, L., AND MCPHEE, N. Landscapeec: Adding geographical structure to cellular evolutionary algorithms. In *Proceedings of Midwest Instruction and Computing Symposium* (2011), MICS '11.

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

[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] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008. (With contributions by J. R. Koza).