Abstract—This study introduces a new representation, landscape automata (LSA), for encoding heightmaps that may be used for terrain generation or other procedural content generation. Landscape automata are evolvable state-conditioned quadtrees with embedded decay parameters. Landscape automata are used to both match idealized landforms and to generate a heightmap with controllable connectivity for agents using the heightmap as terrain. Parameter studies on both mutation rate and number of states in the automata are performed. Mutation rate is found to have a modest impact on performance while the number of states used both has a large impact on fitness and a different type of impact for each of two types of fitness functions. Landscape automata are demonstrated to be well able to match idealized landforms, providing a palette of varied approximations with a variety of secondary features. They are also able to generate heightmaps that, viewed as terrain, form challenging mazes.

I. INTRODUCTION

LANDSCAPE induction, the creation of landscapes with desired properties, is an active area of research in automatic content generation for games. Procedural content generation (PCG) is the generation of game content automatically with computer code. Its goal is to provide automatically generated game content either for immediate use or for hand-off to a designer for polishing and use in a larger framework. In the latter case, it is good if the content generator can be controlled by the game designer and it is also desirable for the content generator to supply a number of choices to the game designer.

This study introduces a new representation, called landscape automata (LSA), for searching the space of heightmaps. Search is performed with an evolutionary algorithm and so is an example of search-based procedural content generation, a variant of PCG which incorporates search rather than trying to write a PCG system that can generate acceptable content in a single pass. A survey and the beginnings of a taxonomy of search-based PCG can be found in [17]. This study demonstrates two different methods of granting a designer control over the heightmaps generated by LSAs. This first permits the designer to specify an idealized land-form, like a hill, crater, or ridge which an evolutionary algorithm then approximates. Both good and mediocre approximations result, providing a palette of landforms for the designer. The second control method treats the heightmap as a maze in which barriers consist of height differentials between adjacent grids of the heightmap that exceed a critical threshold. This method permits the designer to specify checkpoints that must be mutually accessible and permits the designer to then use any of several fitness functions based on characteristics of paths between fitness functions. Many fitness functions of this sort are given in [1]; in this study four checkpoints are used and the algorithm maximized the total pairwise distance.

Though automated level generation in video games can arguably be traced back to the roguelikes of the 1980s (Rogue, Hack, NetHack, . . . ), the task has recently received some research interest. In [13] levels for 2D sidescroller and top-down 2D adventure games are automatically generated using a two population feasible/infeasible evolutionary algorithm. In [16] multiobjective optimization is applied to the task of search-based procedural content generation for real time strategy maps. In [9] cellular automata are used to generate, in real time, cave like levels for use in a roguelike adventure game. Landscape automata can be used to generate landscape features as in [6] or to generate maze like levels, structured as heightmaps.

The work in the current study grows out of work published in both [6] and [1]. The second of these demonstrated four representations for encoding evolvable mazes: a direct encoding, a positive and a negative generative representation, and a representation as a heightmap. In [6] a representation called a midpoint L-system was used to evolve terrain maps with the objective function of matching an idealized crater or hill. In this study we demonstrate that LSA can both approximately match idealized terrain models to provide a palette of terrain features and also create height-map based mazes.

A. Dynamic Programming

Dynamic programming [8] is an ubiquitously useful algorithm. It can be applied to align biological sequences [12], to find the most likely sequence of states in a hidden Markov model that explain an observation [18], or to determine if a word can be generated from a given context free grammar [10]. Dynamic programming works by traversing a network while recording, at each network node, the cost of arriving at that node. When the cost of a new path is not superior to one that is already found, the search is pruned, otherwise the minimum cost of reaching the node is updated and search continues. A number of variants of a dynamic programming algorithm are used in this study, each representing a different fitness function. An example of creating mazes with a dy-
dynamic programming based fitness function, intended as robot path planning algorithms, appeared in [3]. The second type of fitness functions in this study use dynamic programming to calculate the distance between pairs of checkpoints.

The remainder of this study is organized as follows. Section II specifies several representations for evolvable mazes. Section III specifies the experiments performed. Section III-A gives several possible fitness functions that emphasize different qualities in a maze. Section IV gives and discusses the results, contrasting the outcomes of different fitness functions. Section V states conclusions and succinctly summarizes the contributions of the study as well as outlining possible next steps for this line of research.

![Recursive quadrature of the map region](image)

**II. LANDSCAPE AUTOMATA**

Landscape automata are a type of self-driving finite state device that repeatedly partitions a heightmap down to the pixel level while simultaneously generating a heightmap. Partitioning is accompanied by state transitions driven by the identity (quadrant number) of the partition region. Each state contains a height modifier, to be added to the height, when a partitioning yields a transition to the height, and a cascading decay parameter that acts to limit the increase in height as more transitions are made. Figure 1 shows the partitioning process. The regions in a given partitioning are numbered 0, 1, 2, 3 in reading order:

```
0 1
2 3
```

An example of a LSA is given in Figure 2. To generate a heightmap the automata is called at state 0 with an initial height of zero and an initial multiplier of 1.0. The drawing area is then partitioned into quadrants and the automata recursively calls itself in each quadrant, making the appropriate transition for that quadrant. Each chain of downward transitions is separate, maintaining its own local values for both height and the accumulated decay parameter. It might be profitable to view the recursive calls to a landscape automata as being structured by a quadtree[14]. Quadtrees are trees in which each branch is a four-quadrant partition of an image. As the automata enters a state it updates the recursive multiplier value by multiplying it by the decay parameter associated with the state. The automata then adds its height value, e.g. “+4”, multiplied by the recursive decay parameter, to the height along this chain of calls. The recursive calls terminate when the drawing area is the size of a single grid of the final heightmap, in which case the current height value is simply assigned to that grid. A LSA thus specifies a heightmap for a given size of square drawing arena, filling a square array with height values. Notice that an LSA makes no use of random numbers - it is a deterministic specification of a heightmap. LSAs share with their earlier cousins, midpoint L-systems [6], the property that they can be called at less than their full resolution to generate a coarser heightmap suitable for distant views of the object encoded.

Crossover of LSA is performed by treating the list of states as a linear chromosome and subjecting it to two point crossover. A point mutation is accomplished by first picking a state and then picking one of the nine objects (4 height adders, 4 transitions, one decay parameter) and modifying one of them. Height adders and transitions are selected uniformly at random. The state’s height multiplier is modified by adding a uniformly distributed number in the range [-0.1,0.1] and then reflecting any values that wander outside of [0,1] across the appropriate boundary, either \( x \rightarrow -x \) or \( x \rightarrow 2-x \). Height adders take on integer values \{0, 1, 2, 3, 4\} both during initial population generation and mutation. The height multipliers are initialized in the range [0.75,1.0] with the value selected uniformly at random.

**III. DESIGN OF EXPERIMENTS**

The evolutionary algorithm used in this study is steady state [15] using a population of 100 LSA. Selection and replacement are performed with size four single tournament selection. This model of evolution selects four members of the population, copying the two better over the two worst. The list of states of the two copies are subject to two point crossover. Each copy is then subject to \( 1-N \) mutations, with the number chosen uniformly at random, for \( N =4, 7, 11, \) and 15. Two types of fitness functions are used.
The first takes a specialized idealized landscape feature and attempts to minimize root-mean-squared RMS error of the normalized heightmap generated by the LSA with the landform. The second uses a dynamic programming based function to maximize the total distance between each pair of check points placed in the terrain grid. Evolution is continued for 40,000 mating events and each experiment is comprised of 30 independent replicates of the evolutionary algorithm.

A. Fitness Functions

The scale of the heightmaps generated by LSAs are not well controlled so we normalize the heightmaps produced to have a maximum height of one. The idealized landscape features used are shown in Figure 3.

![Fig. 3. The hill, crater, and parabolic ridge functions used as idealized landscape features.](image)

The LSA is evaluated on a 128x128 grid yielding 16384 values for comparison of the function with the heightmap. RMS error is used as the objective function. This objective function is called the *(Hill, Crater, or Parabolic Ridge) landscape match* functions. The equations for the idealized landforms are as follows.

**Hill**

\[ f(x, y) = \frac{1}{1 + x^2 + y^2} \]  
with \(-3 \leq x, y \leq 3\).

**Crater**

\[ g(x, y) = \frac{1}{1 + (9 - x^2 - y^2)^2} \]  
with \(-5 \leq x, y \leq 5\).

**Parabolic Ridge**

\[ h(x, y) = \frac{1}{1 + (3 + y - x^2)^2} \]  
with \(-5 \leq x, y \leq 5\).

The second type of fitness function treats the heightmap as a maze in a manner similar to [2] where passage between two grids of the heightmap is possible if their difference in height is less than a critical value. This study uses the critical height \( H_{crit} = 0.1 \). Recalling that the height is restricted to lie in the range [0,1] this is a fairly steep value: it makes the maze more visible in figures. Four checkpoints, one in the center of each edge of the heightmap are used. The six pairs of distances between these checkpoints are computed and the evolutionary algorithm uses their total as its objective function. If the four checkpoints are not all mutually accessible, the LSA that generated the heightmap is assigned a fitness of zero. This objective function creates a maze with long, winding passages joining the checkpoints. This objective function is called the *Maze* function.

IV. RESULTS AND DISCUSSION

The number of states in an LSA have a significant impact on the quality of solutions. Box plots comparing 10, 30, and 90 state automata for the hill function are shown in Figure 4.

![Fig. 4. Impact on solution quality of varying the number of states in an LSA for the hill function.](image)
largest ranges for numbers of mutations is significant in both cases. The best choice for both the crater and hill functions is thus 90 states and 1-4 mutations; this value was adopted without testing for the parabolic ridge.

Figure 7 shows the impact of changing the number of state in the LSAs for the maze fitness function. The result is different from that for the landform matching experiments. The best average fitness results from using 10 states but the highest fitness individual was located in the 90 state experiment. This suggests that the populations were still improving with 90 state LSAs requiring more evolution to reach their full potential.

Figure 8 shows the same evolved heightmap rendered as both a terrain map and as a maze. The maze rendering, while hiding much detail, directly exhibiting possible and impossible moves between terrain grids. The use of such dual views has the potential to grant a game designer an omniscient abstraction (the maze) while giving the player a much less lucid but more realistic view. Notice the correspondence of features such as the large box canyon in the upper right quadrant of both maps.

A. Qualitative diversity

Figures 9, 10, and 11 show examples of terrain renderings of heightmaps for the hill, crater, and parabolic ridge functions. Note that the different examples for the same idealized landform can look quite different. The encoding as a landscape automata can yield good approximations to the landforms as in the first example of each or it can yield interesting but rough approximations. The second and third landforms in Figure 10 show cascading craters and minor additional craters. The second landform in Figure 11 shows one of the lowest quality results from the file of best-of-run LSAs. These are the result of transitions at lower levels of the recursive process to the state that initiates the formation of the crater. The resulting heightmaps are partially self-similar and, if the recursion were continued indefinitely, would yield a type of fractal representing a generalization of self similar fractals.

The many variations of the theme of the original idealized landform generated by the evolution of LSAs forms a potential strength of the representation. If both gives the designer a compact form of the heightmap he asked for and also suggests intriguing variations of it. This is something that happens often in evolved art - local optima of the fitness function are often of similar desirability to any global optima that are located. Since LSA can only approach continuous, curved surfaces in the limit they must approximate them when a finite number of grids are used, a fact that drives diversity and many local optima approximate the ideal curve.

V. Conclusions and Next Steps

This study demonstrates that LSAs can be used as small objects that store variations on designer-specified landforms. They also serve as a representation for heightmaps that can create terrain with designer-specified connectivity properties. This can be extended with techniques similar to those in [11] to place particular objects like towns or ammunition
Fig. 8. An evolved landscape displayed as a maze - walls represent boundaries too steep to climb - and as an explicit terrain map.
dumps into the landscape with desired relative distances. In [4] individual maps were used as tiles to efficiently design gigantic level maps. These techniques can also be applied to the terrain maps generated with LSAs.

Another property of LSAs not used in the current study is the ability to cascade LSAs. A hill could be studded with craters by simply passing the recursive call from an LSA that builds a hill to one that builds craters. This cascading property can be used to any depth and the height in the quadtree decomposition of the landscape where the hand-off happens controls the size of the subsidiary features.

A. Next Steps

Single parent techniques [7] augment a standard evolutionary algorithm with an additional variation operator called single parent crossover. A selection of structures of the same type as the evolving population, called the ancestor set is added to the algorithm. Whatever crossover operator is used by the algorithm is used, in a one-sided fashion, by single parent crossover to permit population members to cross over with members of the ancestor set. This technique ensures that critical building blocks cannot be lost and provides domain knowledge to the degree that the ancestors are high quality solutions to the problem. While modest care is needed to ensure that the ancestors are not simply cloned via multiple crossover events, the technique can substantially reduce time to solution or improve solution quality.

In [5] it was shown that single parent crossover also has the ability to substantially focus evolutionary search in
the vicinity of the ancestor set. This suggests that single parent techniques would be an excellent next step for LSA. Both improving solution quality and focusing an algorithm near an interesting variation on a landform are potentially desirable. Since they are finite state devices that can encore the functionality in a subset of their genome, LSAs are well suited to single parent crossover.

REFERENCES


