Search-Based Procedural Generation of Maze-Like Levels

Article *in* IEEE Transactions on Computational Intelligence and AI in Games · October 2011

DOI: 10.1109/TCIAIG.2011.2138707 · Source: IEEE Xplore

Daniel Ashlock
University of Guelph
401 PUBLICATIONS **4,853** CITATIONS

Cameron Mcguinness
University of Guelph
13 PUBLICATIONS **230** CITATIONS

Search-Based Procedural Generation of Maze-Like Levels.

Daniel Ashlock and Colin Lee and Cameron McGuinness

***Abstract*—A correctly designed dynamic programming algorithm can be used as a fitness function to permit the evolution of maze-like levels for use in games. This study compares multiple representations for evolvable mazes including direct, as well as positive and negative indirect representations. The first direct representation simply specifies, with a binary gene, which squares of a grid are obstructed. The second paints the maze grid and passage is allowed only between colours that are the same or adjacent in a rainbow. The positive and negative representations are developmental and evolve directions for adding barriers or digging "tunnels". These representations are tested with a design space of fitness functions that automatically generate levels with controllable properties. Fitness function design is the most difficult part of automatic level generation and this study gives a simple framework for designing fitness functions that permits substantial control over the character of the mazes that evolve. This technique relies on using checkpoints within the maze to characterize the connectivity and path lengths within the level. Called *checkpoint based fitness*, these fitness functions are built on a menu of properties that can be rewarded. The choice of which qualities are rewarded, in turn, specifies within broad limits the characteristics of the mazes to be evolved. Three of the representations are found to benefit from a technique called *sparse initialization* in which a maze starts mostly empty and variation operators fill in details while increasing fitness. Different representations are found to produce mazes with very different appearances, even when the same fitness function is used. The example fitness functions designed around dynamic programming with checkpoints are found to permit substantial control over the properties of the evolved mazes.**

I. Introduction

The problem of *level generation* is a key task within the field of Procedural Content Generation (PCG). Its goal is to provide a map for a level in a game together with populating that level with challenges and rewards. This study is focused on automating the first half of this task, map generation. The maps generated in this study are best thought of as mazes and the goal is to create an evolutionary algorithm-based system that gives the user a good deal of control over what types of mazes are being generated. The issues of culs-de-sac, branching and reconvergence of the paths in the maze, and the representation used to encode the mazes are all explored in this study.

This study evolves maps 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 [21]. The different representations and fitness functions tested in this study yield maps with very different appearances from building-like floor plans to intestinal mazes. These various appearances are useful for designing different parts of a game - an above ground structure and a natural cavern have a very different look and feel.

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 [18] levels for 2D sidescroller and top-down 2D adventure games are automatically generated using a two population feasible/infeasible evolutionary algorithm. In [20] multiobjective optimization is applied to the task of search-based procedural content generation for real time strategy maps. In [11] cellular automata are used to generate, in real time, cave like levels for use in a roguelike adventure game.

The work in the current study grows out of work published in [3]. This earlier study generated two sorts of mazes. The first is based on the zones of control of chess pieces, while the second is based on the chromatic progression of the rainbow. The chess puzzles presented, aside from not using a standard sized chess board, are very similar to *chess mazes* introduced by Bruce Albertson as a type of puzzle used to teach chess moves to novice players [1], [2]. The chromatic mazes were used to demonstrate the generality of the technique. Both types of mazes used in the earlier study were made more difficult for a player by the fact that the walls are not directly visible. Chess mazes have barriers defined by where various chess pieces, placed on the board, can attack; the chromatic mazes only permit a player to move between squares whose colours are adjacent on the rainbow. In both cases, the human player must deduce the permitted moves.

One of the fitness functions used in the earlier research computed the length of the shortest path from entrance to exit of the maze. When the fitness function was maximized this had the effect of making the maze as long as possible, given the representation used to encode the maze. Maxima of this fitness function are winding paths that do not branch. This caused the evolved mazes to have a feature that would be undesirable in many circumstances: the mazes generated

Daniel Ashlock is at the Department of Mathematics and Statistics at the University of Guelph in Guelph, Ontario, Canada, N1G 2W1. email: dashlock@uoguelph.ca

Colin Lee is at the Department of Mathematics and Statistics at the University of Guelph in Guelph, Ontario, Canada, N1G 2W1. email: clee04@uoguelph.ca

Cameron McGuinness is at the Department of Mathematics and Statistics at the University of Guelph in Guelph, Ontario, Canada, N1G 2W1. email: cmcguinn@uoguelph.ca

The authors thank the National Science and Engineering Research Council of Canada, the Ontario Council of Graduate Studies, and the Department of Mathematics and Statistics of the University of Guelph for their support of this research.

are trivial, with the only remaining challenge being to deduce the rules for where a player may move. An alternate fitness function, which attempted to minimize the difference between the shortest path from entrance to exit and some targeted value, was also introduced and resulted in mazes which contained some non-trivial features such as branching and culs-de-sac.

The primary contributions of this study are, first, to explore a variety of representations for encoding mazes and, second, to give a framework for designing fitness functions that give the user substantial control over the character of the mazes that evolve.

A. Dynamic Programming

Dynamic programming [10] is an ubiquitously useful algorithm. It can be applied to align biological sequences [14], to find the most likely sequence of states in a hidden Markov model that explain an observation [22], or to determine if a word can be generated from a given context free grammar [12]. 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. Details of these algorithms are given in Section III. In the course of dynamic programming, other variables may be saved and propagated along with the records of shortest paths. An example of creating mazes with a dynamic programming based fitness function, intended as robot path planning algorithms, appeared in [6].

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

II. Representations for Maze Generation

Representation is a key issue in the design of an evolutionary algorithm [17]. Representation is a point at which domain knowledge can be incorporated. Changing representations can, in the best case, result in thousands-fold improvements in performance [9], though usually more modest improvements are obtained. Four representations are explored in this study.

1) A *direct* representation, in which open and blocked squares within a rectangular grid are specified directly as a long, binary gene. This representation will be called the *first* direct representation.
2) A *direct* representation, in which the squares within a grid are assigned colours from the set { red, orange, yellow, green, blue, violet}. These colours are specified directly as a long gene over the alphabet $\{R, O, Y, G, B, V\}$. An agent can move between adjacent squares if they are (i) the same colour or (ii) adjacent in the above ordering. This representation will be called the *second* direct representation or the *chromatic* representation.
3) An *indirect positive* representation in which the chromosome specifies structures that are placed on an empty grid to form the level. In this representation, walls are explicit and rooms and corridors implicit.
4) An *indirect negative* representation in which the chromosome specifies material to remove from a filled grid to form the level. In this representation, rooms and corridors are explicit while walls and barriers are implicit.

In each of these representations the issue of feasibility (can a player traverse the maze at all) is of some concern. In all three non-chromatic representations, the expressed maze amounts to a specification of obstructed and unobstructed squares on a rectangle of width X and height Y. In three of the representations a new technique called *sparse initialization* was used. Sparse initialization places relatively few obstructed squares in the mazes in the initial population. This makes them easy, low-fitness mazes but leaves all members of the initial population feasible. Greater complexity and higher fitness are then discovered incrementally by the variation operators. The details of sparse initialization are given in the descriptions of the individual representations.

With the exception of the mazes using 50×20 grid shown in Figure 11, all the mazes in this study use a 30×30 grid.

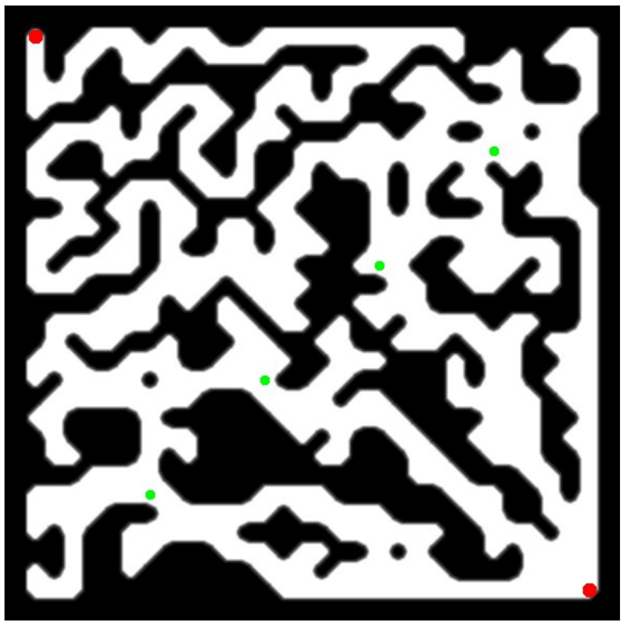


Fig. 1. An example of an evolved maze using the first direct representation. The entrance and exit are marked with large circles, smaller circles indicate checkpoints used by the fitness function.

A. Details of the first direct representation

For an $X \times Y$ board the direct representation is a string of XY bits. Two variation operators are used: uniform crossover with probability p_c and uniform mutation with probability p_m. The uniform crossover operates on two chromosomes, exchanging their bits at each location with independent probability p_c. Uniform mutation operates by flipping the bit at each location with probability p_m. Actual rates for these operators are given in the description of experiments in Section IV. In addition, two experiments are performed using traditional one and two point crossover. An example of a maze evolved with the first direct representation is shown in Figure 1.

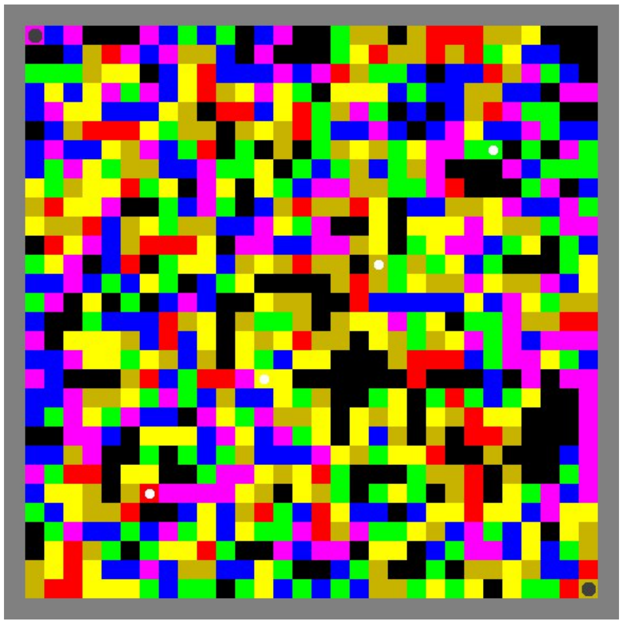


Fig. 2. An example of a chromatic maze. The large black circles mark the entrance and exit, the smaller white dots mark checkpoints.

The probability that a maze in which obstructed and unobstructed squares are equally likely can be traversed is so close to zero that a 500-member initial population, encountered during algorithm development, contained zero traversable mazes. In order to address this problem, a parameter $0 < fill < 1$ was added to the algorithm that is the probability a given square will be obstructed (that a bit in the chromosome will be one). This parameter is set to $fill = 0.05$ in all experiments - a 1/20th full set of initial mazes. The effect of this is to start with traversable mazes of relatively low fitness and permit the variation operators to fill in added obstructions so as to increase fitness in a selection-guided manner. We call this technique *sparse initialization*.

B. Details of the chromatic representation

For an $X \times Y$ board the second direct representation is a string of XY values in the range $0 \leq x \leq 5$ mapping onto $\{R, O, Y, G, B, V\}$. Two variation operators are used: uniform crossover with probability p_c and uniform mutation with probability p_m. The uniform crossover operates on two chromosomes, exchanging their colors at each location with independent probability p_c. Uniform mutation operates by generating a new colour at each location with probability p_m. Actual rates for these operators are given in the description of experiments in Section IV. An example of a maze evolved with the chromatic representation is shown in Figure 2.

Similarly to the first direct representation, a form of sparse initialization is needed with the chromatic representation. The form of sparse initialization used is to initialize the grid to have squares that are green and yellow with those colours being equally likely. A maze initialized in this fashion permits movement between all pairs of squares while making it very easy for mutation to create barriers.

This representation is very different from the others in the following sense. The other representations all specify which squares of a grid are obstructed. The chromatic representation does not directly obstruct any square within the grid: rather it imposes rules about which squares it is possible to move between. This means that it creates implicit walls between squares that define the maze rather than defining it with obstructed squares. Implicit walls create a situation in which the number of bits in a minimal description of a chromatic maze is much larger, at the same grid size, than for the first direct representation. This is because each *pair* of adjacent squares has a potential barrier between them rather than each square being a potential barrier.

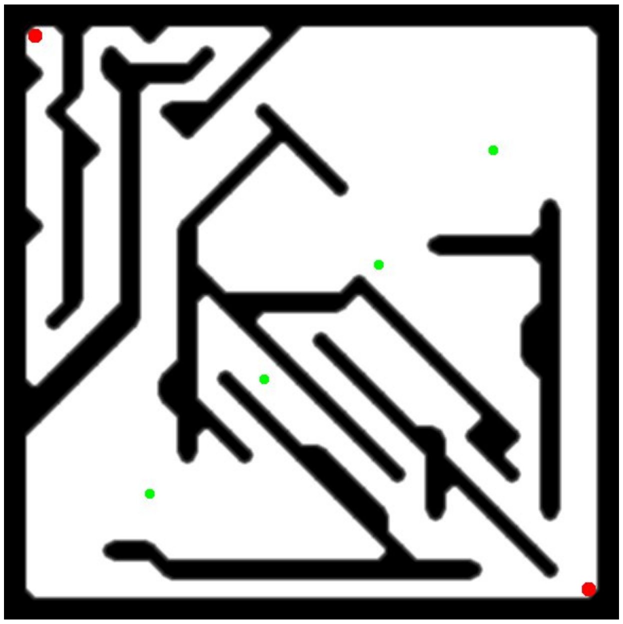


Fig. 3. An example of a positive, indirectly encoded evolved maze. The large red circles mark the entrance and exit while the smaller green ones mark checkpoints.

C. Details of the positive, indirect representation

This representation is similar to the one used in [6]. It is stored as a linear array of integers in the range $0 \leq n \leq 9999$. The integers are used in pairs to specify barriers in an originally empty $X \times Y$ arena with walls at its edge.

The first integer is bit-sliced into a one-bit number, a three-bit number, and a remainder R which is taken modulo the larger of X or Y to yield the length of the barrier. The one-bit number determines if a barrier is penetrating or not. The three bit number is interpreted as a direction for the barrier to run from its starting point W, NW, N, NE, E, SE, S, or SW. The second integer i is split as $x = i\ mod\ X$ and $y = ((i\ div\ X)\ mod\ Y)$ to obtain the starting point (x, y) of the barrier on the grid. The operator div represents integer division without remainder. A penetrating barrier runs from its starting point to its length or the edge of the arena. A non-penetrating barrier stops when it encounters any other barrier. Figure 4 gives an example of unpacking this representation.

There is an additional parameter used in the positive, indirect representation: it is permitted to lay down at most a fixed number Lim of filled squares. Once this number of squares is reached, the remainder of the barrier being laid down, and all other barriers following it, are skipped. Two variation operators are used: uniform crossover with probability p_c and uniform mutation with probability p_m. The uniform crossover operates on two chromosomes, exchanging integers at each location with independent probability p_c. Uniform mutation operates by replacing the integer at each location with a new integer chosen at random in the range $0 \leq n \leq 9999$ with probability p_m. Actual rates for these operators, the number of pairs of integers in a chromosome, and the maximum number of filled squares Lim are given in the description of experiments in Section IV. An example of a maze evolved with the positive representation is shown in Figure 3.

As with the direct representations, there is a very high probability that a maze created with this positive, generative representation will have no path from the entrance to the exit of the maze unless either the number of barriers is kept quite small or barriers start with very short lengths. A maze with few barriers is not likely to be interesting no matter how cleverly they are placed; for this reason initial barriers (but not those created by mutation) have lengths in the range $1 \leq L \leq 3$. This is a third form of sparse initialization.

D. Details of the negative, indirect representation

This representation is stored as a linear array of integers in the range $0 \leq n \leq 9999$. The integers are used in pairs to decide where to remove material, in the form of rectangular rooms and corridors, from an initially filled $X \times Y$ grid. A pair of 4×4 rooms are always placed over the entrance and exit of the maze to increase their "size" in the dynamic programming environment, decreasing the chance a given specification of material removal will fail to join the entrance and exit.

The first integer in a pair is bit-sliced into a two-bit number and a remainder R which is used to determine corridor length or room dimensions. The two-bit number determines the type: skip, north-south corridor, east-west corridor, or room. The length of a corridor is the remainder modulo X or Y, as appropriate. The dimensions of the room are $width = R\ mod\ 4 + 2$ and $height = (R\ div\ 4)\ mod\ 4 + 2$ to yield dimensions for a rectangular room with both side lengths uniformly distributed in the range $2 \leq height, width \leq 5$. The upper left corner (A, B) of the room or corridor are computed from the second integer N by computing $A = N\ mod\ X$ and $B = (N\ div\ X)\ mod\ Y$ where the division is integer division. The "skip" type for objects makes it easy for mutation to *remove* features from a maze. Figure 5 gives and example of unpacking a gene of the sort used by this representation.

Two variation operators are used: uniform crossover with probability p_c and uniform mutation with probability p_m. The uniform crossover operates on two chromosomes, exchanging integers at each location with independent probability p_c. Uniform mutation operates by replacing the integer at each location with a new integer chosen at random in the range $0 \leq n \leq 9999$ with probability p_m. Actual rates for these operators, the number of pairs of integers in a chromosome, and the maximum number of filled squares Lim are given in the description of experiments in Section IV. An example of a maze evolved with the negative, indirect representation is shown in Figure 6.

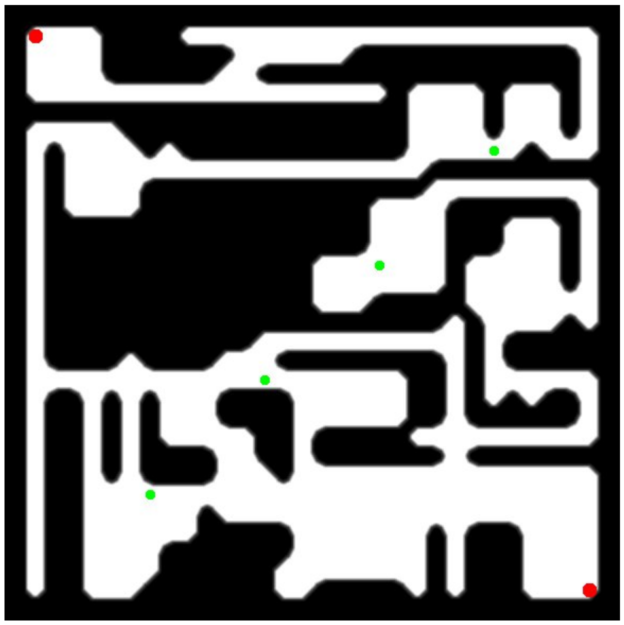


Fig. 6. An example of a negative, indirectly encoded evolved maze. The larger red circles mark the entrance and exit while the smaller green ones mark checkpoints.

The problem of most random mazes being untraversible did not arise with mazes encoded with the negative generative representation to anything like the degree it did with the direct and positive generative representations. For this reason no form of sparse initialization was used with the negative representation.

III. Fitness Function Design

This section specifies elements from which fitness functions for maze-like levels can be built and specifies fitness functions used in the experiments in this study. The key to

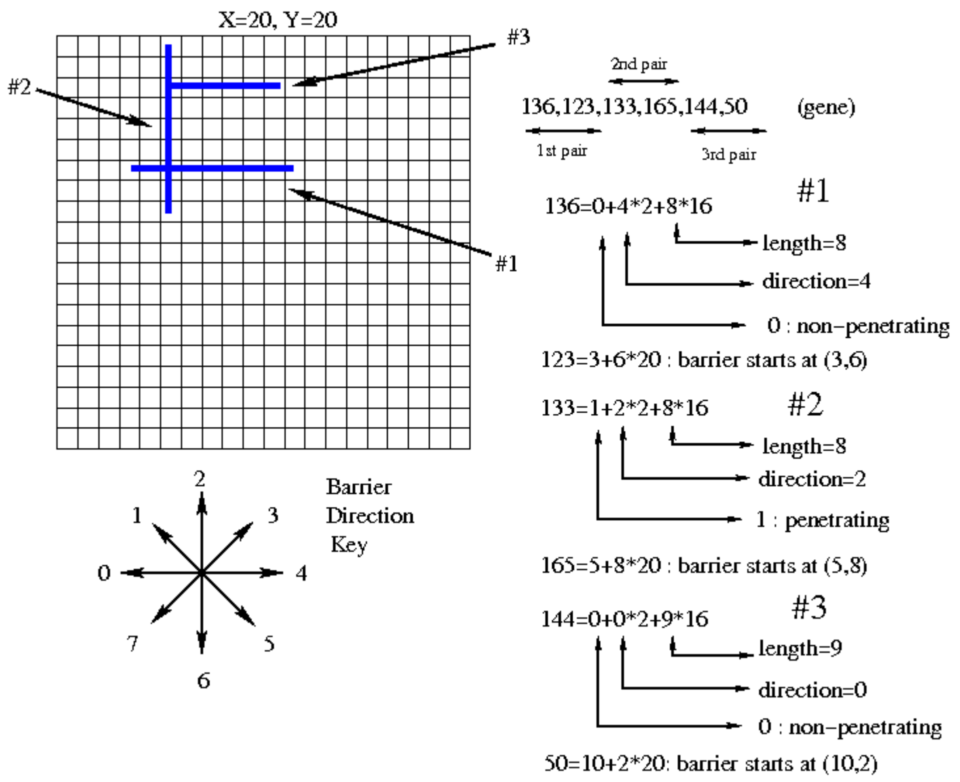


Fig. 4. An example of unpacking a gene for the positive, indirect representation with six numbers (136,123,133,165,144,50) specifying three barriers. The barriers are shown as thick lines running from their beginning grid square to their ending grid square. In practice all squares containing a part of a barrier are registered as "full". Note that barrier #3 stops at barrier #2 after only expressing five of its nominal length of 9 - this is an example of a non-penetrating barrier.

the system presented for constructing fitness functions are the presence of *checkpoints*. A checkpoint is nothing more than a position on the grid used to build the maze. Noting which checkpoints are on a shortest path from entrance to exit permits substantial control over what features of a maze are rewarded by a fitness function. The dynamic programming algorithm notes, for each grid square that is currently the tip of a shortest path, which checkpoint(s) have been encountered along any shortest path from the entrance to that square. When new shortest path (tying the length of an existing path) is discovered leading to a grid square, any checkpoints discovered along that path are also added to the checkpoints recorded for that square. This recording of checkpoints within the dynamic programming algorithm permits membership and the various sorts of reconvergence defined below to be computed.

A. Definitions

The mazes in this study are assumed to have a single entrance square and a single exit square, typically in the upper left and lower right corner, respectively. This constraint is not difficult to relax in practice but yields a far cleaner mathematical environment for exploring fitness functions. Mazes are defined on a rectangular grid of squares in this study.

Definition 1: The entrance (or start) square will be denoted as s, and the exit (or end) square will be denoted as e.

Definition 2: The set of *checkpoints* of a grid is a predetermined subset of the squares of the grid, this set is denoted by $\mathcal{C}$.

Definition 3: The length of a minimal length path between any square x and the entrance square is denoted as $|x|$. This is also known as the *distance* between x and s. If this distance does not exist (because there is no unobstructed path from s to x) we set $|x| = -1$.

Definition 4: A checkpoint is a *member* of a square x if it is on a minimal length path from the entrance square to the square x.

Definition 5: The set of members of a square x is denoted by $\mathcal{M}_x$. The definition of the set of members of x yields a compact method of specifying which checkpoints share a particular relation with a square.

Definition 6: A square x is a *reconvergence* of checkpoints c_i and c_j if both c_i and c_j are members of x. Notice that a reconvergence is a common point along shortest paths through multiple checkpoints.

Definition 7: The *primary reconvergence* of checkpoints c_i and c_j is the smallest path length from the entrance square to any square which is a reconvergence of checkpoints c_i and c_j if a reconvergence exists, otherwise it is 0. The primary reconvergence value is denoted as $prc(c_i, c_j)$. We define this quantity to measure when paths from the entrance through pairs of checkpoints first happen to converge.

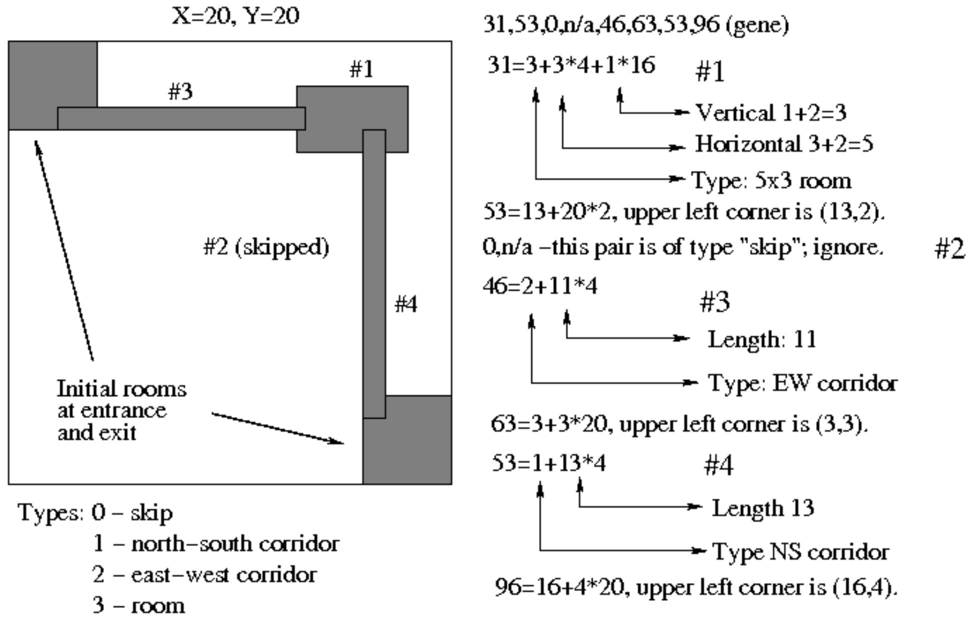


Fig. 5. An example of unpacking a gene for the negative, indirect representation with eight numbers (31,53,0,n/a,46,63,53,96) specifying four objects, one of which is ignored. The number n/a is not specified because it is part of a pair of type "skip". The removed rooms and corridors are shown as discrete, overlapping objects in the figure to illustrate the order of their application. In practice the removed material would be represented as unobstructed grid squares. The 4×4 rooms placed at the entrance and exit are shown in this figure.

Definition 8: Any square y which is a reconvergence of checkpoints c_i and c_j whose distance to the entrance square is equal to the primary reconvergence of c_i and c_j is said to be a *witness to the primary reconvergence* of c_i and c_j.

Definition 9: An *isolated primary reconvergence* of checkpoints c_i and c_j is a witness to the primary reconvergence of c_i and c_j which has no checkpoints other than c_i and c_j as members. The function $iprc(c_i, c_j) = 0$ if no isolated primary reconvergence of c_i and c_j exists and $iprc(c_i, c_j) = prc(c_i, c_j)$ otherwise.

Definition 10: A square x is a *cul-de-sac* if both $|x| \geq 0$ and for all adjacent squares y, $|x| \geq |y|$.

Definition 11: The set of all culs-de-sac in a maze is denoted by $\mathcal{Z}$.

B. Fitness Functions

Assume the fitness function to be maximized is FF. We declare that $FF = 0$ if the entrance or exit squares are obstructed or if the exit square e has fewer than k members. The parameter k, the *mandatory exit checkpoint membership level* gives an interesting form of control over the types of maze that evolve, as we will see in the Sections IV and V. If e has k or more members, then $FF = F_i$ for one of the F_i given below.

- Exit Path Length Fitness

$$F_1 = |e|$$

The earlier study [3] found that this fitness function encourages mazes consisting of a single, long, winding path.

- Primary Reconvergence Sum Fitness.

$$F_2 = \sum_{x,y \in \mathcal{C}} prc(x, y)$$

This fitness function strongly encourages accessibility of all checkpoints with the nature of this accessibility controlled by k. If a checkpoint is not required to be on a shortest path to the exit then higher fitness results from placing it at the end of a cul-de-sac.

- Isolated Primary Reconvergence Sum Fitness

$$F_3 = \sum_{x,y \in \mathcal{C}} iprc(x, y)$$

Isolated reconvergence is harder to achieve than simple reconvergence. It strongly encourages that paths branch, run over checkpoints, and then meet up again later.

- Cul-de-sac Count Fitness

$$F_4 = |\mathcal{Z}|$$

- Cul-de-sac Length Fitness

$$F_5 = \sum_{x \in \mathcal{Z}} |x|$$

IV. Design of Experiments

A very similar evolutionary algorithm was used for all experiments. The algorithm is steady state [19] using size seven single tournament selection. This model of evolution proceeds by mating events that generate pairs of new structures that are reinserted into the population before the next mating event. Size seven single tournament selection chooses seven member of the population without replacement. The two best are copied over the two worst. The binary variation operator is applied to the two copies, then the unary variation operator is applied to each of the copies. The algorithm is run for 500,000 mating events, saving summary fitness statistics for the population every 2,000 mating events. Such a block of 2,000 mating events is called a *generation*. Each experiment consisted of 30 replicates of the evolutionary algorithm performed with distinct random number seeds. The crossover and mutation rates are set to $p_c = 0.05$ and $p_m = 0.01$.

Both of the direct representations and the positive generative representations use a population size of 120 while the negative generative representation uses a population size of 1000. The direct and positive representations used sparse initialization while the negative representation did not: its larger population size makes it very likely that mazes with positive fitness are present in the initial populations for the negative representation.

For all representations both sparse initialization and large populations were tested for their ability to yield an initial population with a majority of feasible mazes. Which of these two options was found to be better was different for different representations. Note that since the negative representation removes material from an initially full grid, sparse initialization would require a more complex structure rather than a simpler one as is the case in the other three representations.

Since our purpose is to make a qualitative comparison of the types of maze evolved with the three representations *rather* than to compare their fitness this rather substantial difference in initialization technique and population size is not a problem: each algorithm was tuned to produce an initial population with solvable mazes *not* to compare the representation's ability to solve the same optimization task.

A. Initial Experiments.

A set of nine initial experiments were performed using fitness functions F_1-F_3 with each of the three non-chromatic representations. These all used a 30×30 grid for the maze with checkpoints set $\mathcal{C} = \{(6,24),(12,18),(18,12),(24,6)\}$. The entrance is square (0,0) while the exit is square (29,29). The mandatory exit checkpoint membership level was set to $k = 3$.

B. Experiments with Culs-de-sac.

A pair of experiments were conducted, using the first direct representation, to test fitness functions F_4 and F_5, which maximize the number or distance from the entrance of culs-de-sac. These experiments use the first direct representation with the same settings and checkpoints as the other experiments that use the first direct representation. As with the other experiments some requirements are placed on the maze: there must be a path from the entrance to the exit and all checkpoints must be in unobstructed squares.

C. Changing the board size.

Three experiments, using identical algorithm settings to the initial experiments for all three non-chromatic representations using fitness function F_3 were performed on a 50×20 grid with checkpoints $\mathcal{C} = \{(5,15),(15,15),(5,35),(15,35)\}$. These experiments are supposed to demonstrate the ability of the algorithm to work for non-square boards and boards of other sizes.

D. Experiments with the Chromatic Representation.

Four experiments were conducted, consisting of 30 replicates each, with the chromatic representation. These experiments differed from one another in using fitness functions F_1, F_2, F_3, and F_4. These experiments use a 30×30 grid and otherwise use the same algorithm settings as the initial nine experiments, except that the required number of checkpoints that must be members of the exit square were reduced to $k = 1$. These experiments are intended to see if the various fitness functions used in this study work well with a representation that specifies barriers implicitly.

E. Verification of Sparse Initialization and Crossover.

The sparse initialization and the choice of a low-rate uniform crossover were both based on preliminary experimentation with a development version of the software created for this study. Four additional experiments were performed, based on the initial experiment with the first direct representation using F_1. Two of these experiments replaces uniform crossover with one-point and two-point crossover. The other two increased the $fill$ parameter from 0.05 to 0.1 and 0.2. Recall that this parameter governs the probability that a square in the first direct representation be obstructed.

V. Results and Discussion

Figure 7 shows a maze from each of the nine original experiments. Up to the limitations imposed by the representation, mazes evolved with fitness function F_1 which rewards only the length of the path from the entrance to the exit produced long winding paths with little or no branching and only a few side corridors. This is true of all thirty replicates for each of the three experiments using F_1. The effect of the k parameter, when this fitness function is used, is to force the checkpoints to be on this path. This means a designer can use checkpoints to shape the path to some degree.

For the mazes generated using F_2, the requirement that three checkpoints be members of the exit comes into play. If none of the checkpoints are required to be on a shortest path to the exit then it is very likely to have one checkpoint on such a path and place the others at the end of long culs-de-sac. This maximizes the reconvergence numbers for pairs of checkpoints very well. When we require, by setting $k > 1$, that several checkpoints be on a path from the entrance to the exit then the algorithm placed the required number of checkpoints on paths from entrance to the exit and places remaining checkpoints at the end of culs-de-sac. In all three of the mazes shown in Figure 7 the lower left checkpoint is at the end of a long cul-de-sac. The mazes evolved with the negative and first direct representation place the checkpoints in a branching structure while the positive representation places all three on a long, winding path.

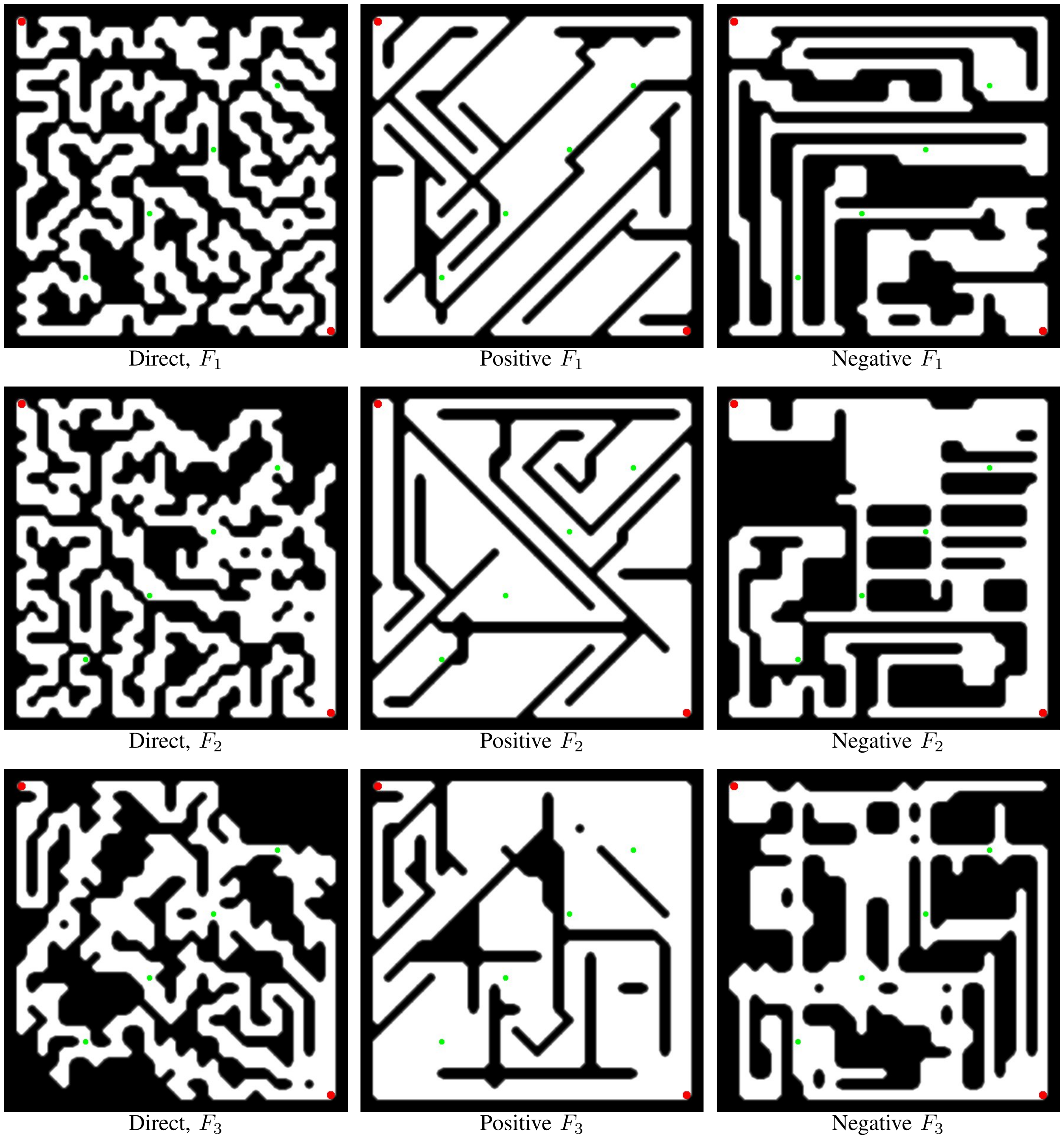


Fig. 7. Shown is a maze from each of the nine initial experiments. The mazes are organized so that all mazes in a column use the same representation and all mazes in a row use the same fitness function. The large, red circles mark the entrance and exit while the small, green dots mark the checkpoints. These mazes are 30×30 and the entrance, at (0,0) is in the upper left corner while the exit, at (29,29) is in the lower right.

All three of the mazes evolved with F_3 place all four checkpoints on distinct paths to the exit. The pattern of branching varies somewhat and both generative representations produce more additional large branches. This fitness function was the most likely, during algorithm development, to produce entire populations that are all of zero fitness. It also is not as strongly impacted by the value of k as the need to produce unique reconvergences to obtain fitness forces checkpoints to be on direct paths from the entrance to the exit. A checkpoint in a cul-de-sac can give rise to *at most* one unique reconvergence while one along a path from entrance to exit can have a unique reconvergence with every other checkpoint.

The different population sizes chosen for experiments with different representations were the result of initial experimentation with the goal of finding a population size that permitted the algorithm to generate acceptable results. In [8] it is shown that, for some problems, very small populations sizes are superior while others function better with large populations. The relationship between final quality and initial population size is complex, obscure, and representation dependent. Since the goal of this study is to show what different representations and fitness functions can do, we did not perform an extensive parameter study to locate good initial population sizes and rather used a quick *ad hoc* set of trials.

The first direct representation produces almost "intestinal" patterns that are more reminiscent of a natural cave than the results produced by the two indirect representations. The positive representation generated the sparsest maps as well as those that looked like intentional or planned structures. The negative representation produced mazes with a distinct character from the other two, but hard to describe stylistically. Examination of the full range of evolved mazes shows that the negative representation was the best at placing checkpoints on distinct paths from the entrance to the exit; the direct representation was second best by this criterion; the positive representation came in a distant third in this regard.

In the initial stages of the research reported in [6], a version of the direct representation was tried and failed horribly: no path from entrance to exit was found in almost all mazes in the initial population. In this study the use of sparse initialization (in both the direct and positive representations) solved this problem. In this case *sparse initialization* means initialization to a state of the maze where relatively few squares are obstructed. The decision to use low-rate uniform crossover in this study was made before sparse initialization was implemented, as a means of enhancing the heritability of feasible mazes. When this choice was later revisited for the first direct representation in a final experiment, reported in section V-D, we find that the performance of standard crossover is similar to but significantly superior to the uniform crossover used in most of the experiments.

The use of sparse initialization is interesting to consider from the perspective of a fitness landscape as well. Sparse initialization was used to create a situation in which there are a very large number of different paths, on average, from entrance to exit and from either entrance or exit to each of the checkpoints. This means that minimal path length and all the reconvergence scores are fairly small. In essence we intentionally create solutions that are very likely to be (i) low fitness and (ii) feasible. This means that we are relying on the variation operators, guided by selection, to climb the hills of the adaptive landscape. Since high fitness mazes are typically very close, when distance is measured in mutations, to infeasible mazes this is probably an effective strategy. Examine the maze in Figure 7 for the direct representation and F_1 (which simply maximizes the path length from entrance to exit). Inspection shows a *majority* of the unobstructed locations will block the path from entrance to exit if they are filled in.

If we look at the mazes generated with F_3, which encourages large, isolated primary reconvergences, then while no single loci in the gene will *block* the path from entrance to exit. Inspection shows that several loci can zero out as many as three of the isolated reconvergences if they are mutated. This means that for F_3, even though mutations to zero fitness are uncommon, mutations to much lower fitness are common in high fitness, nonsparse structures. We thus see that sparse initialization also places chromosomes in a part of the fitness landscape that is far less rugged than the regions containing high fitness structures.

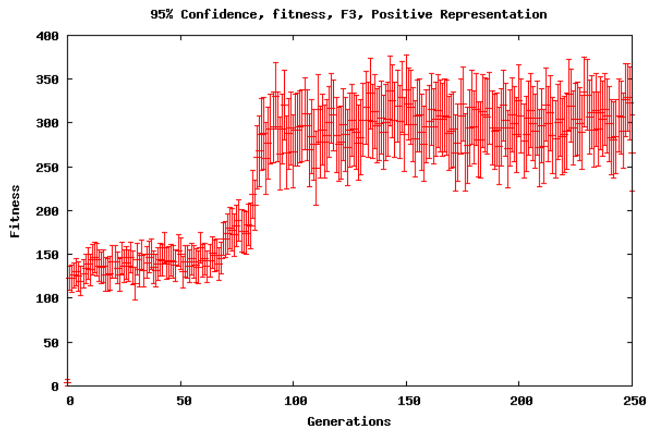


Fig. 9. Mean fitness over the course of evolution, in the first evolutionary replicate, using fitness function $F3$ with the positive representation. Shown with confidence intervals representing a 95% confidence interval.

Examining Figure 8 we see that all nine sets of experiments experienced substantial optimization of the fitness function: verifying this is one of the reasons for using a mean-over-replicates fitness plot of this type. Several of these plots, particularly the one using F_3 and the negative representation, suggest that there is still an upward trend. This is not a great concern; the algorithm can easily be run for a longer time. Of greater importance is the fact that it is not clear that the global optima of any of these fitness functions is the most desirable maze or level design. These algorithms are supposed to provide a broad selection of mazes and the fitness functions are rough heuristics, not clear statements of entirely desirable objectives. Figure 9 shows the behavior of population average fitness in a single run. Notice that the major improvements are via a type of innovative leap (recall a "generation" consists of 2000 mating events of steady state evolution) rather than steady progress. In addition the increase in the width of the confidence intervals on mean fitness as mean fitness improves supports the earlier assertion that high fitness parts of the fitness landscape are mutationally near to cliffs. The innovation represents climbing such a cliff while the high variation in mean fitness represents individuals that have fallen off such cliffs.

A. *Experiments with Culs-de-sac*

Figure 10 shows an example maze evolved with each of F_4 and F_5. The F_4 maze optimizes the number of culs-de-sac in the maze. This, together with the requirement that the checkpoints appear in the area accessible from the entrance, yields a maze with a large number of loops and side passages relative to those produced with the other fitness functions. The function F_5 sums the size (distance from the entrance)

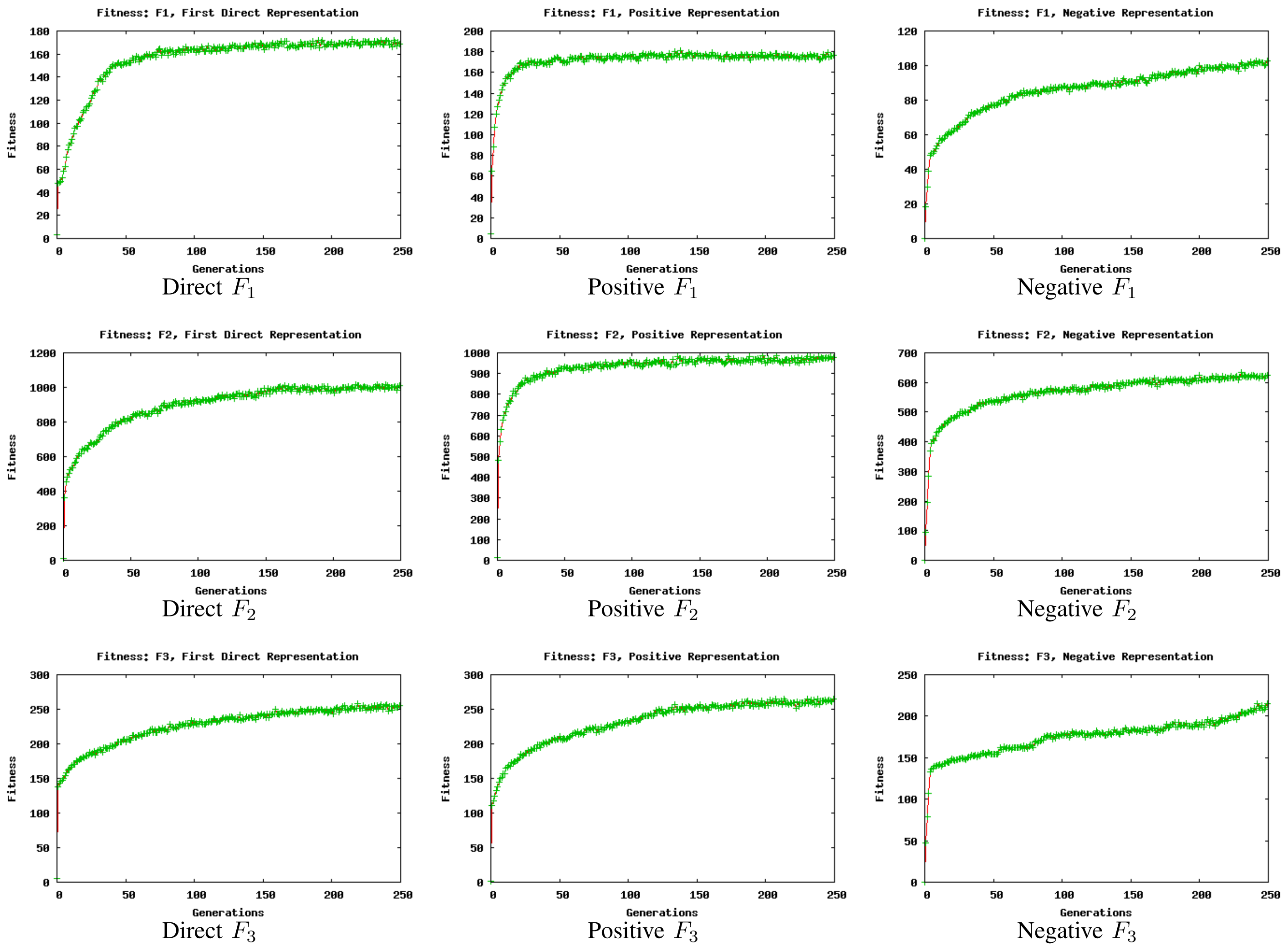


Fig. 8. This figure shows the mean fitness, over all thirty replicates, for each of the three representations and fitness functions used in the initial experiments. The mazes are organized so that all mazes in a column use the same representation and all mazes in a row use the same fitness function.

of all the culs-de-sac. It produces a maze similar to those produced by F_1, maximizing the length of the path from entrance to exit, but with more and longer side passages.

The F_4 mazes are, upon cursory inspection, the most complex and diverse of the sets of mazes evolved with the various fitness functions presented. This suggests that investigation of fitness functions that count culs-de-sac is worth more attention.

B. *Experiments with Different Board Sizes*

These experiments represented a check on the ability of the algorithm, for all three non-chromatic representations, to work on a different board size and with different arrangements of checkpoints. Examples of mazes for each representation are shown in Figure 11. A visual inspection of the 30 mazes produced in each experiment (data not shown) shows that the algorithm is slightly more likely to place checkpoints at the end of long culs-de-sac when the board has a 5:2 aspect ratio but otherwise the results were similar to those obtained for the 30×30 grid.

C. *Chromatic mazes*

Figure 12 shows an example of a chromatic maze and its key from each of the four experiments. The initialization to green and yellow colours is most visible in the mazes evolved with F_4. The other fitness functions all yield a fairly even distribution of colours.

The resulting mazes have similar characters to those evolved with the other three representations. The maze evolved with F_1 is very long but can be solved by mere persistence; it has few side branches and no real choices other than forward and back. The mazes created with F_2 and F_3 place checkpoints not required to be on a shortest path to the exit at the end of long culs-de-sac. The mazes created with F_3 have more branches than those created with F_2.

The mazes created with F_4, which rewards having culs-de-sac, creates a maze with the largest number of branches, closed loops, and reconverging paths. This is also consistent with the behavior of F_4 in the experiments run with the other representations.

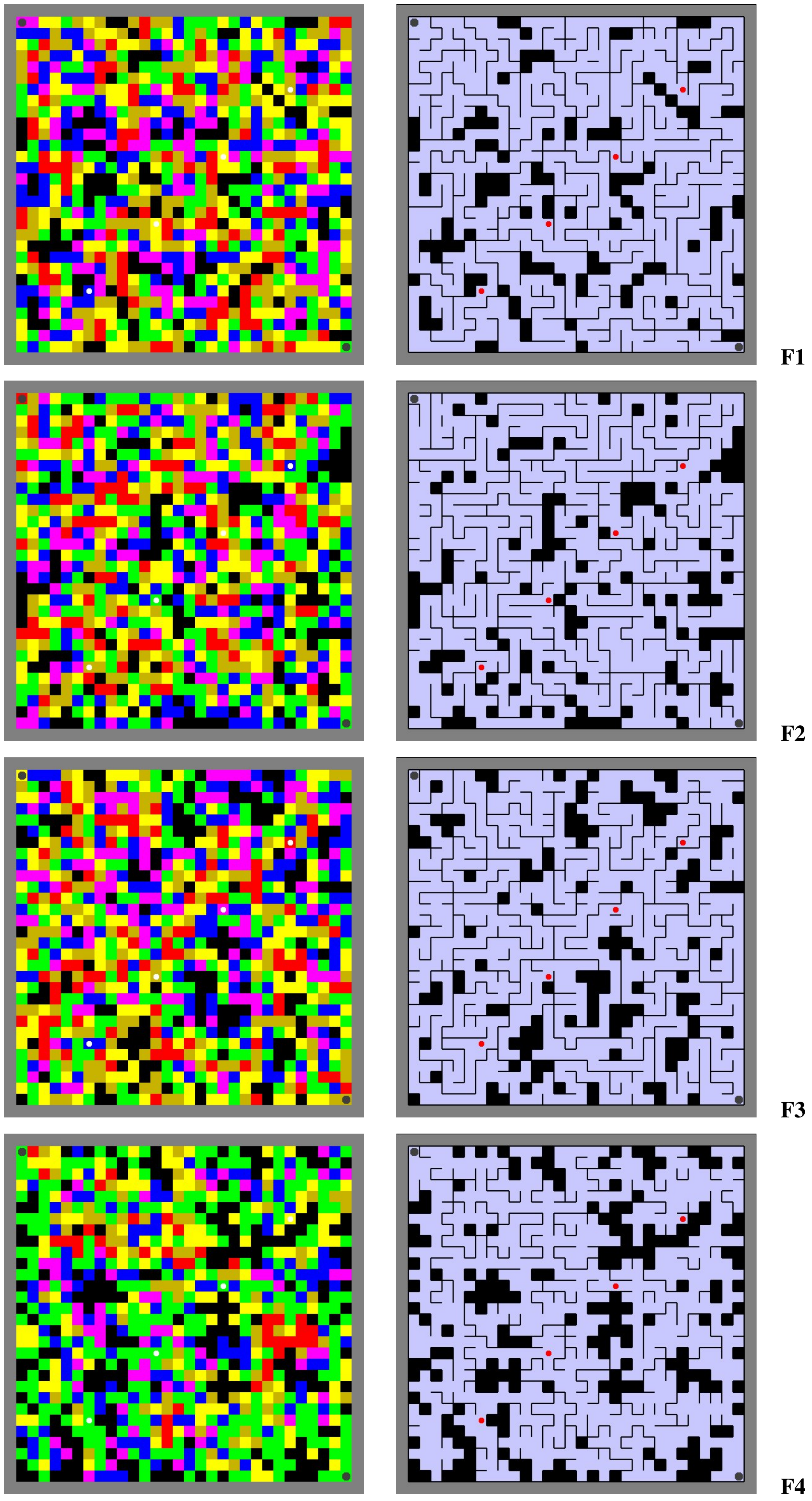
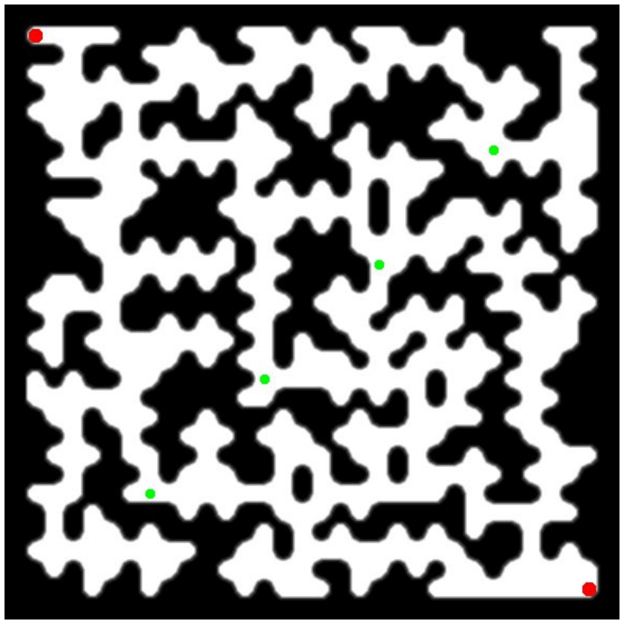
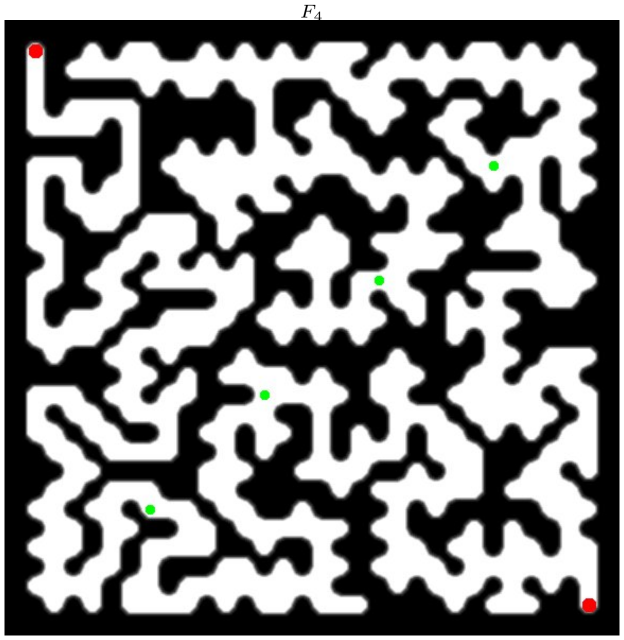


Fig. 12. Chromatic mazes and keys. Black squares are inaccessible from the entrance. Smaller red circles denote checkpoints, larger grey ones the entrance and exit of the maze. The fitness function used to evolve each maze is given to the lower right of its key rendering.



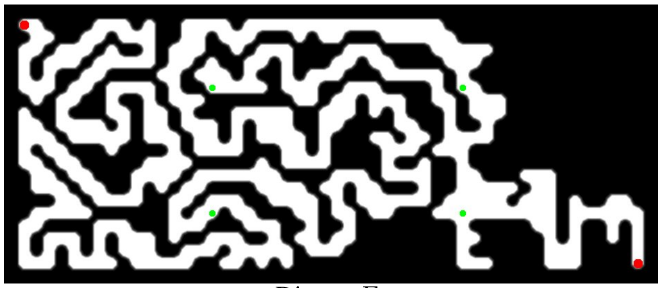
F_4



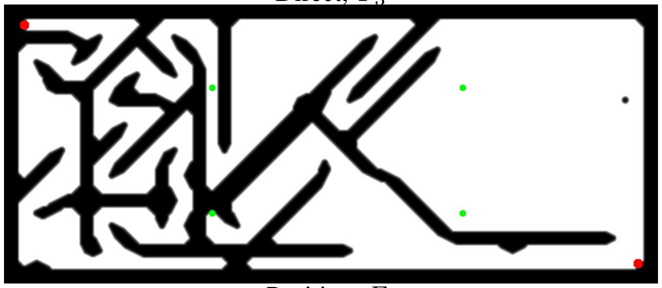
F_5

Fig. 10. Example mazes for fitness functions F_4 and F_5 using the first direct representation. The algorithm that produced these mazes also requires that all four checkpoints appear in the part of the maze accessible from the entrance. The large red circles mark the entrance and exit while the small green ones denote checkpoints.

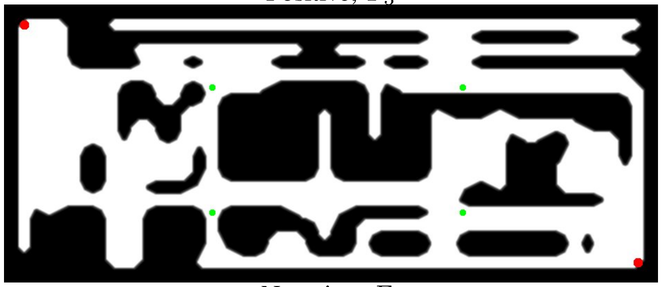
Contrasting the chromatic mazes, rendered directly, with their keys shows that the implicit character of the barriers between squares makes this type of maze much harder to solve than one where the barriers are clearly visible. The chromatic representation is one of a large collection of possible representations for implicitly specified mazes. Chess mazes, such as those evolved in [3], are another implicit maze specification. The experiments in this study show that the space of fitness functions defined here can be applied to implicitly specified mazes.



Direct, F_3



Positive, F_3



Negative, F_3

Fig. 11. Examples of 50 × 20 mazes for all three non-chromatic representations. Large red circles mark the entrance and exit while small green ones mark the checkpoints.

D. Sparse Initialization and Choice of Crossover Operator

Table I and Figure 13 show the results of varying the crossover operator for the first direct representation using F_1. The initial choice of the low rate uniform crossover was made while experimenting with the problem of initial populations with zero fitness. One point crossover exhibits slightly higher performance than uniform crossover, but the difference is not significant. Two point crossover outperformed both uniform and one-point crossover significantly. The margin of improvement is small, $14.4/362.5 \cong 4.0\%$. Figure 13 shows that none of the different crossover operators were particularly faster at reaching the final fitness level. This suggests that the choice of crossover operator is not important, but that future work with these representations and similar representations may wish to use two-point crossover.

The results of varying the $fill$ parameter were more dramatic. When the fill parameter was set to 0.05, the standard setting, all 30 replicates produces a maze with positive fitness. When $fill = 0.1$ 17 of 30 replicates produced a feasible maze. When $fill = 0.2$ none of the 30 replicates produced a feasible maze. This result strongly supports the use of sparse initialization for the first direct representation. It also tends to support its use based on early experimentation

Crossover Operator	Fitness	Confidence Interval
Uniform	362.5 ± 4.0	(358.5,366.5)
One-Point	365.1 ± 2.5	(362.6,367.6)
Two-point	376.9 ± 2.6	(374.3,379.5)

TABLE I
SHOWN ARE 95% CONFIDENCE INTERVALS FOR EXPERIMENTS WITH THE FIRST DIRECT REPRESENTATION USING F_1 WITH DIFFERENT CROSSOVER OPERATORS.

with the chromatic representation and the indirect, positive representation.

It might be natural to ask why not just set $fill = 0$, starting with an empty grid and letting the variation operators do all the work. One of the factors that permits evolutionary algorithms to function is the random initialization. This initialization and the subsequent shake-out of the population acts to select the basin of attraction of the dynamical system represented by the evolutionary algorithm in which a given replicate arrives at its final solution. Even a modest amount of randomization, such as that represented by a setting of $fill = 0.05$ is likely to substantially increase the diversity of mazes located by the EA.

E. Algorithm speed

Running a set of 30 replicates for 500,000 mating events requires about as much time as a coffee break (8-20 minutes). The actual timing is somewhat variable, depending on the fitness function used. Oddly this is *not* because the different fitness functions require different amounts of time to compute. The current version of the algorithm computes all the elements required by any of the fitness functions and then performs a trivial computation to return the exact fitness function being tested. This was done to permit rapid exploration of the space of fitness functions and to leave a hook for future research on multicriteria optimization based versions of this software. The bottleneck in the fitness function is running the dynamic programming algorithm. The mazes of the sort that get a high score from F_4 require that far more squares be visited (with repetition for multiple paths from the entrance to a square) than the mazes that evolve under the influence of F_1. This phenomena is most apparent when sparse representations are used: dynamic programming runs *very* slowly on an almost empty maze. If the researcher watches the trace of a run, the rate of data reports jumps upward as the fitness increases.

The current version of the algorithm is more than adequate for producing huge libraries of mazes with similar properties but different details. Optimizing to compute only the factors needed in a given fitness function will yield a small increase in speed, but the real point to look for speed improvement is the dynamic programming algorithm. The algorithms used in this study were designed to make explorations of the space of fitness functions easy rather than being optimized

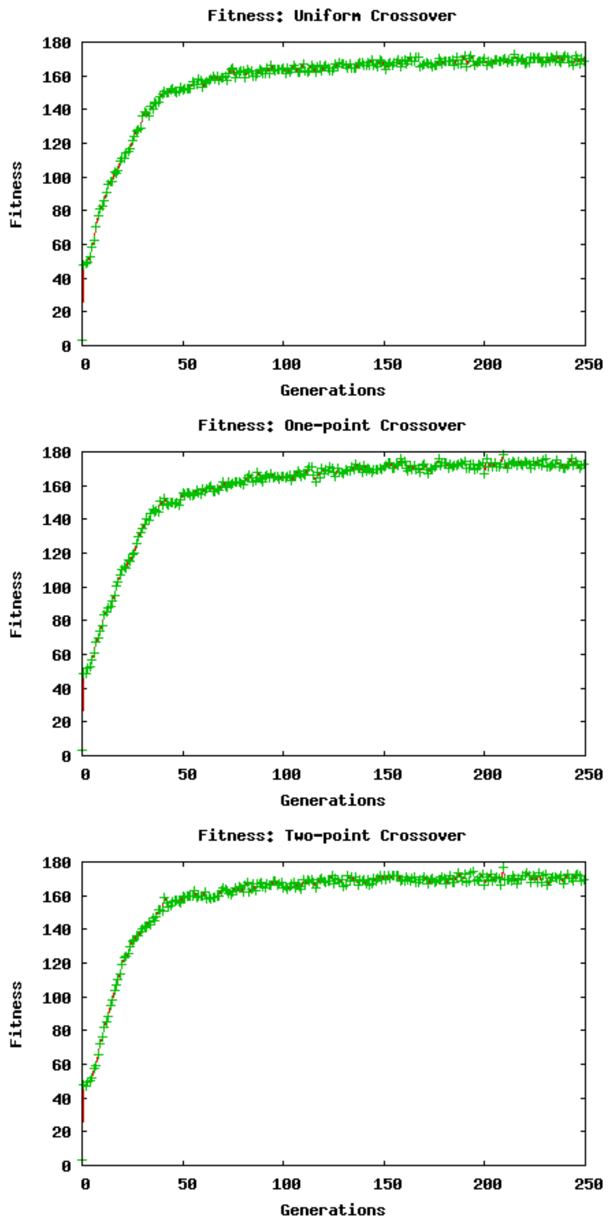


Fig. 13. Mean fitness over thirty replicates for uniform, one-point, and two-point crossover, as a function of number of generations. A generation consists of 2,000 mating events.

for speed. Substantial research exists on optimizing dynamic programming algorithms and the algorithm used here can be parallelized trivially by running one copy of the evolutionary algorithm on each processor. In addition, useable mazes arise long before 500,000 mating events. Optimizing the effort/return tradeoff of evolution is another area where additional speed may be obtained.

F. Fitness landscapes and sparse initialization

Both direct representations and the positive indirect representation used a form of sparse initialization. The effect of this type of initialization is to create mazes with very few obstructions. A maze with few obstructions is likely, for all five fitness functions, to have low positive fitness. In particular, zero fitness because a lack of any path from the entrance to exit is avoided by sparse initialization. This, in turn, makes the variation operators bear the brunt of the burden of generating high fitness mazes. The experiments show that they do this effectively. The experiments in which the $fill$ parameter was varied show that sparse initialization is necessary.

Examine Figure 9. This figure shows that, in one run, as the mean fitness of the population increases so does the variance of the fitness. Examining the evolved mazes shown, it is obvious that there are many squares which, if obstructed, will cause a catastrophic decrease in fitness. These two pieces of evidence demonstrate that high fitness mazes are likely to have mutants with much lower fitness. Colloquially, the heights of the fitness landscape have many cliffs. The sudden increase in fitness shown in Figure 9 is probably driven by a sudden jump *up* one of these cliffs. This "high fitness implies instability" may be an example of a type of self-organized criticality. It also provides a reason, in addition to *ad-hoc* observation of unfit initial populations and the experiments varying $fill$ with the first direct representation, why sparse initialization may be a good idea.

VI. Conclusions

The primary contributions of this study are threefold. First, the study demonstrates that four different representations can be used, with the same fitness functions, to generate maze like levels for use in games. The representations generate mazes with very different appearances and characters.

Second, this study defines several elements, computable with a simple dynamic programming algorithm, that can be used to build a large number of different fitness functions. Five such fitness functions are tested in this study, and many others can be built from primary reconvergence, isolated primary reconvergence, path length from entrance to exit, number of culs-de-sac, path lengths of culs-de-sac, number of checkpoints that are accessible, and number of checkpoints on shortest paths from the entrance to the exit of the maze. The study also demonstrates that the mazes generated with different fitness functions are substantially different from one another in terms of branching factors, loops within the maze, and the placement and length of culs-de-sac.

Third, the study demonstrates that the fitness functions yield comparable results on all four representations tested, including the chromatic representation which specifies a maze implicitly rather than explicitly.

The key rendering of the chromatic representation is a tool for apprehending the connectivity and solubility of the implicitly represented maze in an explicit form. The used of multiple renderings, some of which are available only to a designer, is a potentially rich area. Processing the data from the dynamic programming algorithm in various ways could yield many views useful to a designer.

Sparse initialization, used in all the representations except the negative indirect representation, is a simple but potentially valuable technique for use in the implementation of the representations presented in this study and, potentially, others. Its value was directly verified for one representation and fitness function and is suggested by results during algorithm development for the other two representations where it is used.

While exact quantization of the optimized speed of procedural content generation via the methods prototyped in this study remains to be done, the study demonstrates that procedural generation of a variety of different types of maze like maps can be performed rapidly. For off line generation of libraries of content, the techniques in this study are already beyond the speed required for application. Real time content generation is certainly within the realm of the possible once the algorithms have been optimized.

VII. Next Steps

The most obvious next step for this research is to build a GUI tool for designing maze-like game levels. In this context the algorithm would act as a designer's assistant. A dialog box for piecing together a fitness function out of the elements described here would permit a designer a great deal of freedom to control the types of maze that evolve. The number of fitness functions *not* explored in this study is immense. As researchers found new elements these could be added to the list of options. A palette of representations, including but not limited to those presented here, would increase the flexibility of the tool.

A clear direction in which to extend this research it to find other fitness function elements that could be used to increase the reach of the techniques. Elements that are easy to compute from the variables present in the dynamic programming algorithm used in this study would be most desirable. One such element would be the number of inaccessible squares in a maze, either in absolute terms, or that are nominally unobstructed but not accessible from the entrance of the maze.

The use of checkpoints, while key to defining many of the fitness function elements used, was not extensively explored in this study. The mazes in this study used four checkpoints and only two different arrangements of those checkpoints were tested. Varying the number and arrangement of checkpoints should yield a great deal of control over the type of mazes that arise, however that is beyond the scope of this initial study It is expected that there will be some unexpected consequences of moving checkpoints and varying the number of checkpoints used. Likewise changing the k parameter, denoting the number of checkpoints that must be a member of the exit square, requires more exploration.

While several representation were tested in this study, many others are possible. Grammatical systems, particularly L-systems [13], [16] seem a natural candidate. We are aware of no research generating mazes with L-systems. Our group has worked with L-systems for other purposes [4], [5], [7] and our intuition is that the sort of constraint handling performed with dynamic programming would be difficult to achieve with L-systems. Nevertheless L-system remain a technology of interest, in particular because a successful L-system representation would generate levels with remarkable speed. We note that Parish and Muller [15] have used L-systems to design road networks, a similar task to maze design.

Another factor that would work well with the dynamic programming based fitness functions presented in this study is the incorporation of measures of area filled. At its simplest, this could consists of noting how many squares of a maze grid were filled in. More sophisticated measures could test for the presence of open areas of at least a given shape or of a particular size.

Another obvious venue, not treated in the current paper, is the use of multicriteria optimization. Given that we generate multiple fitness elements it is obvious to attempt simultaneous optimization of multiple criteria. A maze that had a long path from entrance to exit but also maximized the sum of its two largest isolated reconvergence numbers would have a different character from the mazes shown in this study. A maze with many culs-de-sac and a high sum-of-reconvergences is another that intuition suggests would be interesting.

One method for designing large maps is to build tiles and then snap the tiles together on a square or hexagonal lattice. If a set of openings on the boundary of a tile are defined, then the techniques used in this paper would be a natural tile-generating engine. Even simple fitness functions such as sum of path lengths between pairs of boundary openings would yield complex tiles as long as the number of openings is at least three.

It is also possible to create a far more complex type of structure with a simple variation of this technique. Imagine we have a level where each square has a different level. If we have two agents, one of which can leap up or down only a height of ten feet and the other of which can leap up or down twenty feet. Then we have, in a given array of heights, two distinct maze connectivities. Using F_1 on the maze with ten foot connectivity and F_3 on the maze with twenty foot connectivity and then applying multicriteria optimization to both functions would yield a maze in which the second agent would have far greater freedom of movement.

In requiring, explicitly or implicitly, that the checkpoints appear within the accessible portion of the maze, we suggest a more general technique: required features. Suppose that portions of the grid are pre-defined and immutable, such as a central cavern or rooms that must be included in particular position on the map. Such fixed features can be imposed on the grid before a given maze-specifying representation is used. If checkpoints are placed inside these features, maze properties relative to those features can be controlled as in the work presented here. Such a system permits a designer to specify basic features of the maze and then let the evolutionary algorithm "fill in the details" to complete a level. The fitness function can be written with desired details, e.g. a long distance between two particular features, in mind.

References

[1] B. Albertson. *Chess Mazes: A New Kind of Puzzle for Everyone*. ChessCafe.com, South Chatham, MD, 2004.

[2] B. Albertson. *Chess Mazes 2*. ChessCafe.com, South Chatham, MD, 2008.

[3] D. Ashlock. Automatic generation of game elements via evolution. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence in Games*, pages 289–296, Piscataway NJ, 2010. IEEE Press.

[4] D. Ashlock, K. M. Bryden, and S. P. Gent. Evolutionary control of bracked L-system interpretation. In *Intellegent Engineering Systems Through Artificial Neural Networks*, volume 14, pages 271–276, 2004.

[5] D. Ashlock, K. M. Bryden, and S. P. Gent. Creating spatially constrained virtual plants using L-systems. In *Smart Engineering System Design: Neural Networks, Evolutionary Programming, and Artificial Life*, pages 185–192. ASME Press, 2005.

[6] D. Ashlock, T. Manikas, and K. Ashenayi. Evolving a diverse collection of robot path planning problems. In *Proceedings of the 2006 Congress On Evolutionary Computation*, pages 6728–6735. IEEE Press, Piscataway NJ, 2006.

[7] D. A. Ashlock, S. P. Gent, and K. M. Bryden. Evolution of L-systems for compact virtual landscape generation. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 3, pages 2760–2767. IEEE Press, 2005.

[8] W. Ashlock. Using very small population sizes in genetic programming. In *Proceedings of the 2006 Congress on Evolutionary Computation*, pages 319–326, 2006.

[9] W. Ashlock and D. Ashlock. Single parent genetic programming. In *Proceedings of the 2005 Congress on Evolutionary Computation*, volume 2, pages 1172–1179, Piscataway NJ, 2005. IEEE Press.

[10] R. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[11] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, New York, NY, USA, 2010. ACM.

[12] D. E. Knuth. *The Art of Computer Programming Volume 2: Seminumerical Algorithms*. Addison-Wesley, New York, NY, 1997.

[13] A. Lindenmayer. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology*, 16:280–315, 1968.

[14] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 3(48):44353, 1970.

[15] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM.

[16] P. Prusinkiewicz, A. Lindenmayer, and J. S. Hanan. *The algorithmic beauty of plants*. Springer-Verlag, New York, 1990.

[17] F. Rothlauf. Representations for evolutionary algorithms. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 2613–2638, New York, NY, USA, 2008. ACM.

[18] N. Sorenson and P. Pasquier. Towards a generic framework for automated video game level creation. In *Proceedings of the European Conference on Applications of Evolutionary Computation (EvoApplications)*, volume 6024, pages 130–139. Springer LNCS, 2010.

[19] G. Syswerda. A study of reproduction in generational and steady state genetic algorithms. In *Foundations of Genetic Algorithms*, pages 94–101. Morgan Kaufmann, 1991.

[20] J. Togelius, M. Preuss, and G. N. Yannakakis. Towards multiobjective procedural map generation. In *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–8, New York, NY, USA, 2010. ACM.

[21] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation. In *Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 141–150. Springer Berlin / Heidelberg, 2010.

[22] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2), 1967.