Automatic Generation of Game Elements via Evolution.

Daniel Ashlock

Abstract— **This study presents a system for automatically producing puzzles for use in game design. The system incorporates an evolutionary algorithm that optimizes the puzzle to a specified level of difficulty. The fitness function uses dynamic programming to compute the minimum number of moves required to solve a puzzle. Two types of puzzle are explored, one is a maze based on chess pieces and the other uses colors as related by the color wheel to create an implicit maze. The evolutionary algorithm is able to produce a wide variety of puzzles at specified levels of difficulty. The algorithm can thus be used to provides a library for game design or for variable game content. The technique is flexible and can be generalized to puzzles of remarkable complexity by simply upgrading the dynamic programming algorithm used in the fitness function. It is found that puzzles requiring a maximum number of moves to solve are, potentially, less difficult because such puzzles present the player with few choices. This problem is addressed by modifying the algorithm to search for puzzles with a smaller minimum number of moves required, leaving more room in the puzzle for choice and its attendant confusion.**

I. Introduction

This study provides a proof-of-concept for using an evolutionary algorithm as a puzzle generator for automatic creation of game content. Evolutionary algorithms are capable of locating a diverse collection of optima in a complex function [5] and so are a natural technology for creating a large collection of distinct puzzles for use in game design.

After a search for past art, relatively few similar efforts were located. The chess puzzles presented in this study, aside from not using a standard 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 [2], [3]. Related uses of evolutionary computation and artificial intelligence include the following. Multi-objective evolutionary computation has been used to create opponents in games such as driving agents [1] and to design combinatorial games [7] with the additional objective of finding playable games. Artificial intelligence has been applied to the problem of balancing board games by modifying their rules dynamically [11] and to the problem of dynamically adjusting the difficulty of a video game [9]. In [15] the authors use an evolutionary algorithm to select rules for a game. None of these papers seeks to create a palette of game elements similar to those produced in this study.

The technique introduced here, while used only on variations of two types of puzzles, is easy to generalize to many other types of puzzles. Working with a new puzzle type requiring only that dynamic programming code for the type of puzzle in question be written. In addition, a representation must be constructed to enable the use of evolutionary computation. The evolutionary computation required is neither difficult nor sophisticated, rather the problem solved has immediate application in simplifying the process of game design by creating large collections of puzzles of variable hardness, rapidly. One strength of the method is that it produces useful results even if it only samples local optima of the search space. Global optimality is not a requirement for locating diverse collections of puzzles.

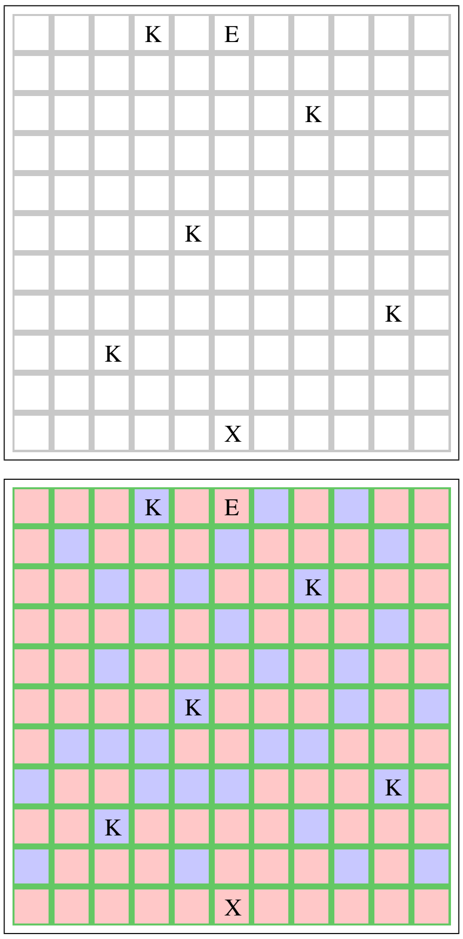


Fig. 1. An example of an evolved chess puzzle for a Rook agent, using five knights, that requires 23 moves to traverse. The entrance grid is designated by **E** and the exit by **X**. The upper panel shows the puzzle as it appears to a player, the lower makes obstructed squares in light blue and open squares in pink. This differential shading makes solutions much easier to see.

Dynamic programming [6] is an ubiquitously useful algo-

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

The author would like to thank the National Science and Engineering Research Council of Canada for its support of this research.

rithm. It can be applied to align biological sequences [13], to find the most likely sequence of states in a hidden Markov model that explain an observation [16], or to determine if a word can be generated from a given context free grammar [10]. The version of dynamic programming used in this study 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 that already found, the search is pruned; otherwise the minimum cost of reaching the node is updated and search continues. Details of the dynamic programming algorithms used in this study are given in Section III-A.

The remainder of this study is organized as follows. Section II gives a specification of the puzzles that will be generated with evolutionary algorithms. Section III gives the design of the evolutionary algorithm and fitness functions as well as specifying which experiments are performed. Section IV gives and discusses the results. Section V draws conclusions and outlines possible next steps for this research.

II. Specification of Puzzles

An almost infinite variety of types of puzzles can be specified by creating rules for safe movement in an otherwise hazardous environment. The simplest version of this is a maze in which movement is "safe" if space is not obstructed by the walls of a maze. The puzzles specified in this study have *implicit* rather than explicit barriers, depending on cultural knowledge, with a game agent presumably taking some sort of damage when they violate the implicit constraints. The first type of puzzle involves knowledge of the moves of chess pieces while the second involves knowledge of the proximity of colors on the color wheel.

A. Chess Mazes

A *Chess maze* is specified by the placement, on a grid, of some number of chess pieces. These pieces do not move, rather the move the chess piece usually makes designate the grids that the piece *covers*. The agent that is traversing the puzzle is assigned a chess piece as well. The agent has a designated entry and exit grid from the maze. The agent's goal is to traverse the maze, via a sequence of moves valid for his assigned chess piece, without ever traversing a covered or occupied square. An example of an evolved chess maze is given in Figure 1, shown both as the player might see it (top) and with the covered and occupied squares differentially shaded (bottom). The implicit specification of the maze makes it more difficult to see a solution, a fact that can be verified by working the Example in Figure 1 first using the top specification and second using the bottom. It is presumed that if the agent attempting to traverse the maze enters a covered square or makes a move impossible to his assigned chess piece that he is slain or severely injured. The actual consequences of moving incorrectly for the player's agent type or passing through a covered square is left for the game designer. This difficulty of a maze is measured by the minimum number of moves required to traverse it.

B. Chromatic Puzzles

The chromatic puzzle is created by assigning the colors **R**(ed), **O**(range), **Y**(ellow), **G**(reen), **B**(lue), and **V**(iolet) to the squares of a grid. Safe moves consist of stepping from a color to the same color or one adjacent to it on the color wheel. The difficulty of a puzzle is again determined by the number of moves required to traverse the puzzle. A secondary factor that can be computed is the number of safe moves that require a change of colors. The more color changes that are required the less visible the maze is as a monochrome set of squares. We do not compute this factor here as the use of six colors, in combination with the puzzle evolution strategy, yields relatively few monochromatic multi-square fragments in the solutions to the evolved puzzles. Figure 2 gives an example of an evolved chromatic puzzle.

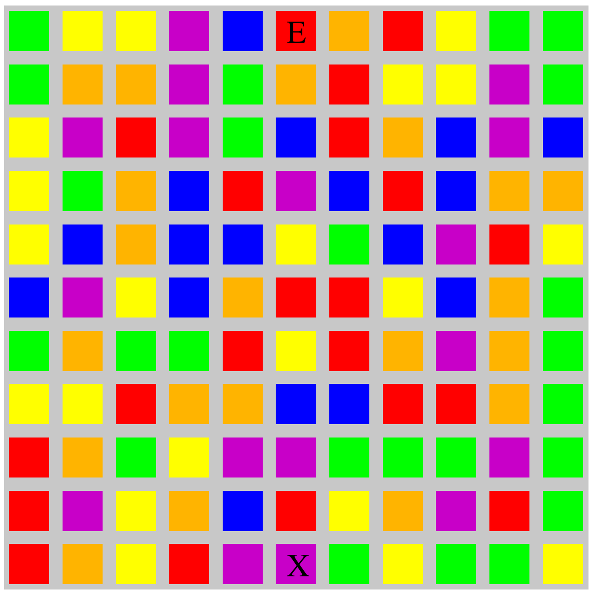


Fig. 2. An example of an evolved 11x11 chromatic maze. The entry (E) and exit (X) are marked with letters. This example requires 94 moves for safe traversal of the maze, the highest fitness found in the 11×11 chromatic maze experiment in which number of steps were maximized.

III. Experimental Design

The evolutionary algorithm used is similar for both sorts of puzzles. A dynamic programming algorithm is used to compute the minimum number of moves required to traverse the puzzle. Details are given in Section III-A. This number of moves is to be maximized in the initial experiments used to set the evolutionary algorithm parameters. Later the alternate function f^*

$$f^*(gene) = C - |f(gene) - target|, \qquad (1)$$

where f is the original fitness function used to compute the fitness of $gene$, was used to bias evolution towards boards that required a specific number of moves. Notice that f^* takes on its maximal value when the value of the original fitness function equals the target. The constant C is chosen large enough to force all results to be positive; its value is otherwise unimportant.

The evolutionary algorithms used in this study are steady-state [14], using size seven single tournament selection [5] as its model of evolution. This model of evolution selects seven population members uniformly at random. The members of this tournament are sorted by fitness and the two most fit are selected for reproduction with their children replacing the two worst members of the tournament.

Except for the 21x21 chromatic puzzles, evolution continues for 40,000 mating events. Evolutionary runs performed for the 21x21 chromatic problem still exhibit an upward fitness trend after 40,000 mating events and so these experiments instead used 100,000 mating events. During reproduction, crossover and mutation are applied to the two selected individuals to produce two children. The variation operators are not the same for the two types of puzzles.

The chess maze is stored as a list of positions for the pieces being placed. This list is considered unordered. A type of uniform crossover is used on this data structure. The positions of each piece specified for a given experiment are distributed uniformly at random from parents to children. Children are rejected (and crossover is re-performed) if two chess pieces are placed in the same square. Mutation operates by moving a piece for the placement to a new, unoccupied location.

The chromatic puzzles store the colors assigned to squares on the board in a linear string of concatenated rows. The length of the gene is the number of squares on the board. The variation operators used are two-point crossover of the string of colors and mutation. Mutation picks a square and changes its color to one selected uniformly at random; this color may be the original color. The chromatic puzzle locating algorithm is thus an entirely standard evolutionary algorithm.

A parameter study was conducted on chess mazes with a rook type agent that compared population sizes 12, 120, 1200, and maximum number of mutations 1 and 3. The number of mutations performed is selected uniformly at random between 1 and the maximum. The study was conducted for three sets of opposing chess pieces: seven pawns, five knights, and a mixed set containing a rook, a knight, and three pawns. Notice that a rook, queen, or bishop can cover a large number of squares and effectively block the board unless pawns or knights are present to reduce their coverage by getting in their way. For each parameter choice and set of pieces, thirty replicates of the evolutionary algorithm were run.

A similar parameter study was run for 11×11 chromatic puzzles (data not shown) which yielded very similar results to the study for chess mazes. Based on the results of the parameter studies, all experiments other than those in the parameter studies were run with population size $P = 1200$ and maximum number of mutations $M = 1$. Experiments for chromatic puzzles of size 6×6, 11×11, and 21×21 were run.

In order to demonstrate that the algorithm can find boards requiring a specified minimum number of moves, a set of runs were performed using the fitness function given in Equation 1. These were set to locate a five-knight chess maze for a rook type agent that required a minimum of eight moves and an 11×11 chromatics maze requiring 40 moves. The maximum fitnesses located for these puzzles were 13 and 94 respectively, making 8 and 40 entirely reasonable numbers. Requesting a number of moves below the maximum possible yields puzzles with more dead ends and branches. This increase in local complexity will, hopefully, yield more interesting puzzles.

A. *Specification of Fitness Functions*

The fitness function for chess puzzles is given below. Recall that the *agent type* is one of pawn, knight, bishop, rook, queen, or king, and that the agent is required to move in the fashion of the chess piece that is its type. We note the pawn is not a good choice of agent type as it leads to really boring chess mazes. The dynamic programming matrix has an entry for each position on the board that records the minimum number of steps along any path examined *so far* to reach each position in the board. The value ∞ is used to denote a square that has not yet been visited.

Chess Maze Fitness

```
Input: Chess piece placement,
       entrance and exit location,
       agent type
Output: Minimum number of moves from
       entrance to exit.
Details:
Place specified chessmen on the board.
Compute all covered and occupied squares.
Initialize dynamic programming matrix to $\infty$.
Call recursive traversal algorithm (RTA)
     with zero moves and entrance position.
Report value in dynamic programming matrix
     at exit position.

RTA(position,moves):
If moves is not smaller than the current value in the
   dynamic programming matrix at position, return.
Otherwise record moves as the new value in the dynamic
   programming matrix at position.
Generate all feasible agent moves from position.
For each move generate the resulting new position.
Call RTA(new position,moves+1).
End For
```

The chromatic puzzle fitness function is the same except that the placement of colors on the board determine which moves are feasible, replacing the placement of chessmen, and all feasible moves are to adjacent squares with permitted colors.

IV. Results and Discussion

The parameter study performed for chess mazes is summarized in Figure 3. The study shows a preference for the largest population size and a mild preference for the lower mutation rate. In all cases, however, the highest fitness value found was located for all parameter settings. For the better parameter settings, the upper quartile marker, maximum, and mode were identical (yielding the boxes with no median bar). The parameter study demonstrates relative insensitivity of the system to the parameters studied.

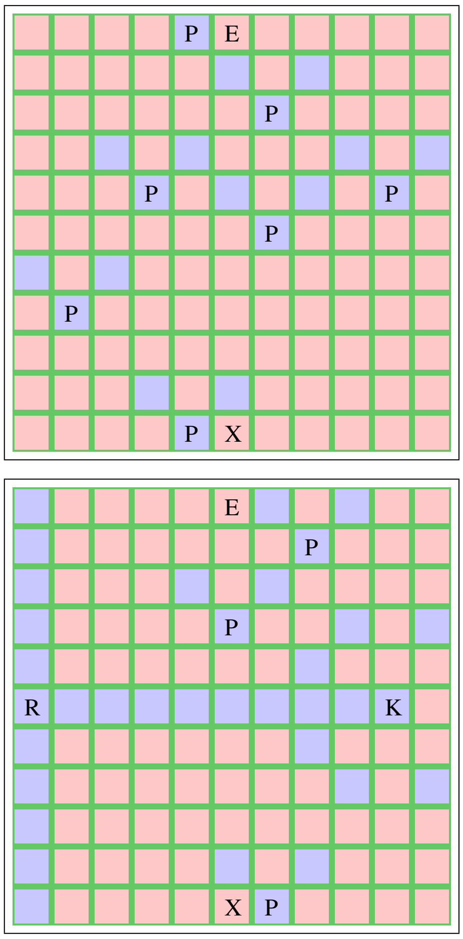


Fig. 4. Example of evolved chess puzzles for a Rook agent. Covered squares are differentially shaded to permit hand verification of fitness. Both examples are maximal fitness among the evolved configurations. The upper configuration uses seven pawns and requires 13 moves, the lower uses a rook, a knight, and three pawns and requires 15 moves.

An example of an evolved board for the experiments placing five knights to challenge a rook-type agent appears earlier in Figure 1. Examples of evolved puzzles for the seven pawns and mixed rook, knight, and three pawns puzzles appear in Figure 4. Notice that in the lower panel of Figure 4 the rook and knight are in the same row with the knight close to one wall and no intervening pawns. Variations of this configuration appear in 22 of the 30 replicates performed for these chessmen. Traversing the resulting bottleneck around the knight takes 5-7 rook moves and so is an excellent feature for a maze that seeks to maximize rook moves. Another feature common to both sets of pieces with pawns in them is a pawn either adjacent to or covering a square adjacent to the entrance and exit of the maze, forcing 1-2 added moves. Both these features demonstrate that evolution is functioning efficiently to make a challenging maze.

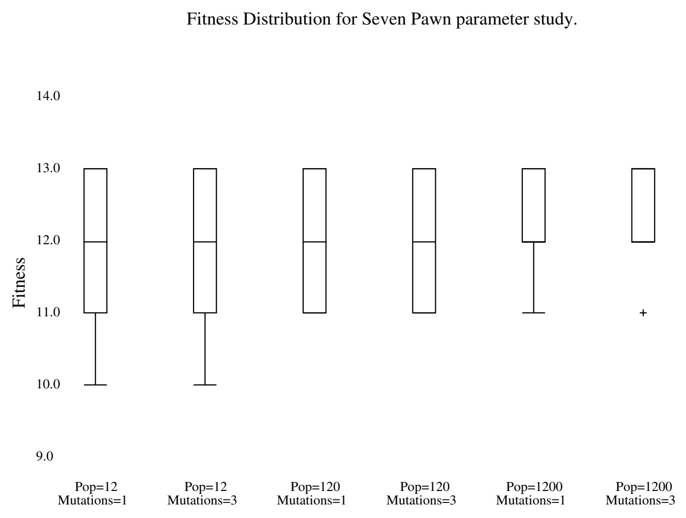
The runtime of the algorithm for chess puzzles is strongly dependent on both the choice of chessmen deployed and the amount of time evolution has been running. Chess mazes requiring a very large minimum number of moves to solve have very little choice of route. This means they may be easier than their raw fitness numbers suggest but it also means that, after the first few hundred mating events, the dynamic programming algorithm runs much quicker. Comparing the five knights and seven pawns experiments, the five knights problems evolved noticeably faster. Examining the solutions, it is easy to see that the seven pawns problem leaves far more moves available to a rook type agent. Since each of these moves represents a potential alternate route, the dynamic programming has more work to do on the seven pawns problem.

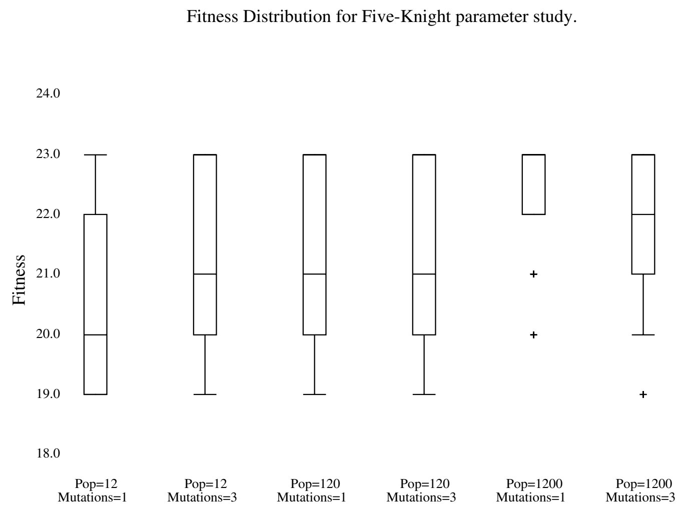
One goal of this study is to demonstrate that the algorithm can locate diverse collections of puzzles. In none of the experiments was the same puzzle located twice. The results of the 30 replicates for the 6×6 chromatic mazes are shown in Figure 5. This results suggests that, even without niching or similar diversity promoting measures, the approach can be used to locate many distinct puzzles of a given type. Runtime for the algorithm is from a second to a few tens of seconds per puzzle located, with the exception of the 21×21 chromatic mazes which require about three minutes per run. The most fit evolved 21×21 puzzle is shown in Figure 6.

A. *Targeted Fitness Experiments*

The two experiments that used the modified fitness function given in Equation 1 set a target of eight moves for a five-knights/rook agent problem and of 40 for and 11×11 chromatic puzzle. All thirty replicates in each of the targeted-fitness runs achieved the target fitness. Examples of evolved solutions to each of these appear in Figure 7. The five-knight experiments with a target fitness of eight ran substantially slower than those allowed to maximize fitness. This supports the hypothesis that the boards with an intermediate number of moves required to solve them might have more blind alleys and wrong turns simply because most of the maze does not consist of a single path.

Generally in a combinatorial space the number of solutions for an extreme value of a parameter, such as minimum number of moves required in a solution to a maze, is far smaller than the number for an intermediate value of the same parameter. This means that it should be easy to find solutions to both sorts of puzzles in this study requiring smaller numbers of move than the maximum found. The 100% success rate in the targeted fitness confirms that this intuition is correct here.





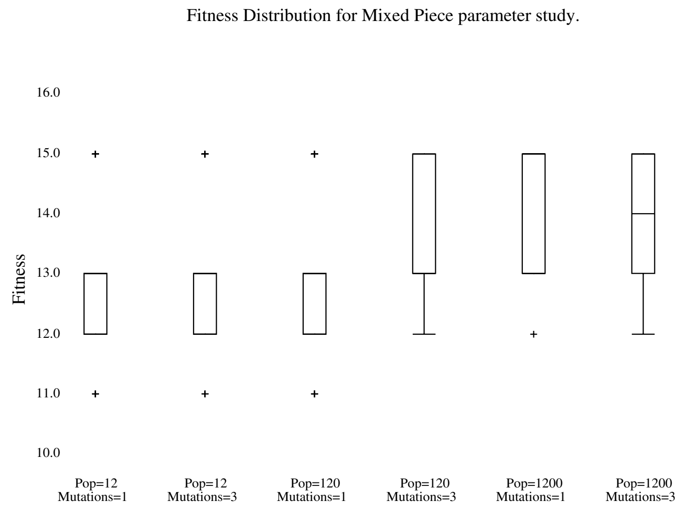


Fig. 3. Boxplots showing fitness distribution for the three parameters studies conducted with a rook type agent. The studies experiments placed seven pawns, five knights, and a mixed set of three pawns, a knight, and a rook. The maximum number of mutations and population size were varied.

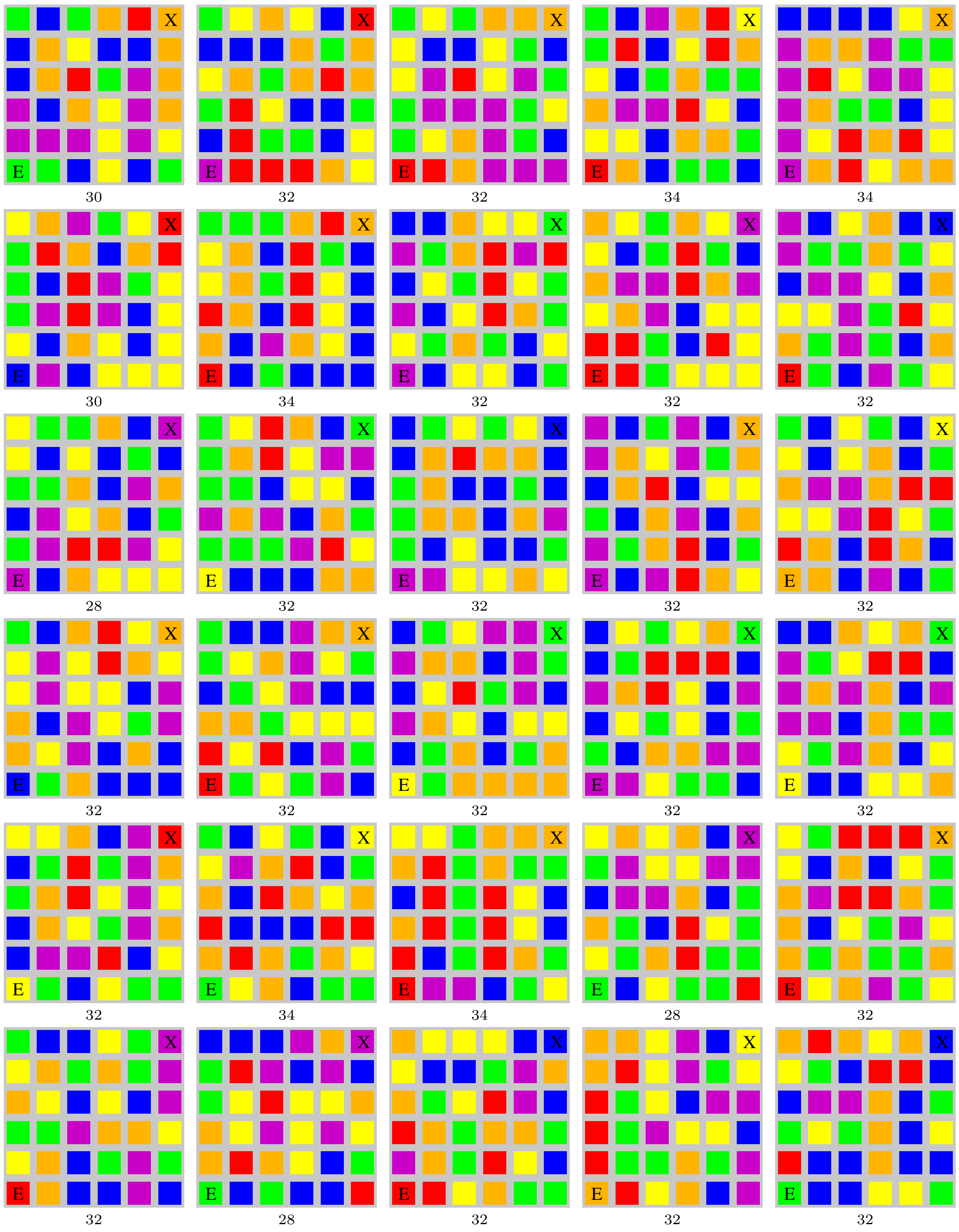


Fig. 5. Most fit results from all 30 replicates in the 6x6 chromatic maze experiment. The minimum number of steps needed to traverse the maze is given below each maze. (E)ntrance and e(X)it squares are marked with letters.

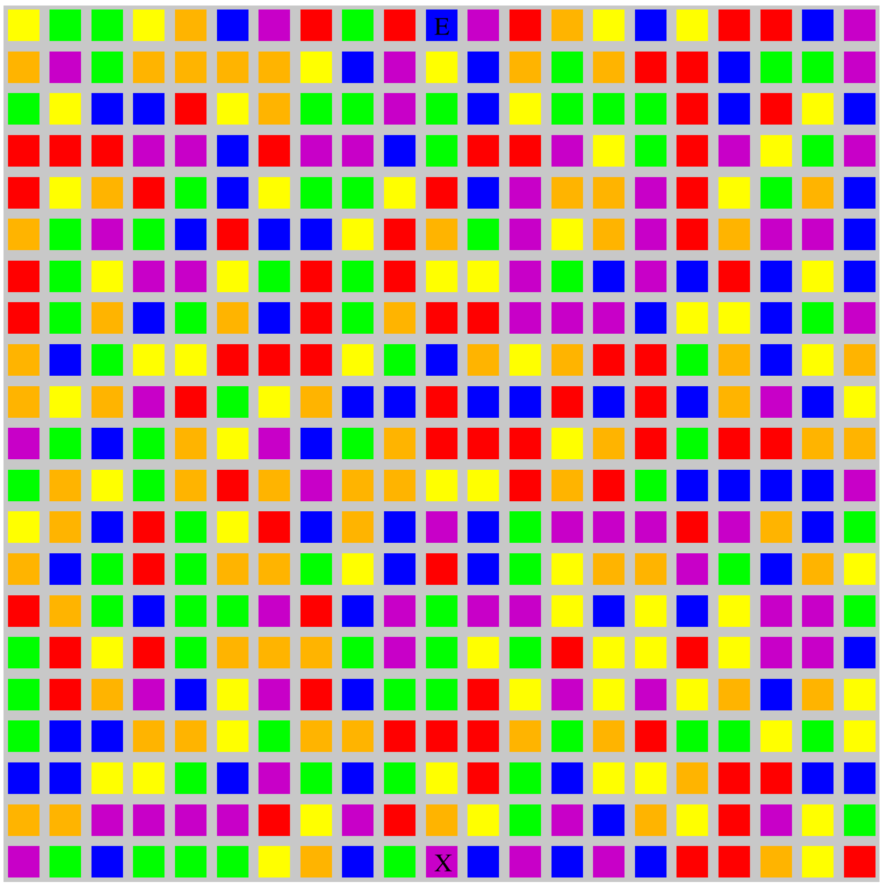


Fig. 6. The most fit of the 21×21 chromatic mazes evolved, requiring 292 moves to safely traverse.

B. Duplicate Solutions

The zero rate of duplication of solutions across seven sets of experiments (those where the final configurations were examined, i.e. not the parameter study) each consisting of 30 replicates was achieved without any of the many possible diversity promoting measures. These techniques include measures such as niching [12] or the various sorts of spatially structured evolutionary algorithms [8]. The ability of the algorithm to essentially "succeed without trying" at achieving diversity is probably because of the size of the sample space. The seven pawns problem, recalling that a pawn may not sit on the entrance or exit or cover the entrance, has $\binom{117}{7} \cong 4.96 \times 10^{10}$ configurations in the search space. For the 11×11 chromatic mazes there are $6^{121} \cong 1.43 \times 10^{94}$ elements in the search space. This means that if the fitness function has many modes, duplications are simply unlikely.

The evolutionary algorithms used in this study are not complex or difficult, they break no new theoretical ground in evolutionary computation. They to some degree resemble those used to create diverse collections of robot planning problems in [4]. The novel contribution is the application of simple evolutionary computation with a dynamic-programming based fitness function to the problem of puzzle construction. The study demonstrates proof-of-concept for using evolutionary algorithms to locate a large number of puzzles *of varying difficulty* rapidly.

V. Conclusions and Next Steps

This study applied a relatively straightforward evolutionary algorithm to generate examples of six different types of puzzles in two categories: chess mazes and chromatic puzzles. The puzzles are all examples of mazes that enhance problem difficulty by using implicit barriers, ones that must be deduced by the agent traversing the maze. These puzzles are intended to supply content for games, in the form of mini-games or in-game puzzles. The ability of an evolutionary algorithm to find large numbers of optima of a complex function means that the evolutionary algorithms presented here can generate libraries of puzzles quickly.

The details of how the puzzles would be integrated into a game are left to the game designer. If a player is unfamiliar with chess moves, or does not know that the color wheel is the critical clue designating safe moves, then they may make some missteps. If making an unsafe move is not deadly but merely does some damage to the player the puzzle can be softened so that a player can discover the rules of the maze by experimenting. It is also not clear that a game designer

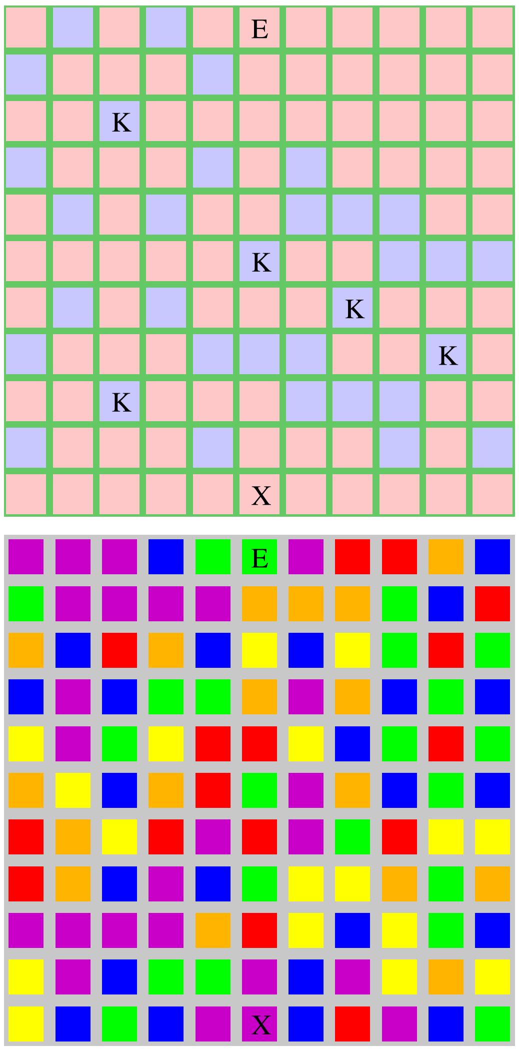


Fig. 7. Exemplary results from the targeted-moves experiments. The upper panel shows a five-knights chess maze intended for a rook agent with the squares differentially shaded. The lower panel shows an 11 × 11 chromatic maze requiring 40 moves.

would want to use the two types of puzzles presented here. This is another reason that this paper should be considered as a proof-of-concept paper.

The technique developed in this study can be applied to implicit mazes of almost any type. It could even be used to generate level or structure layouts by adapting techniques like those in [4]. The technique also could be applied to far more challenging structures. As an example, consider the following generalization of the chromatic maze. The actual maze is generated in three dimensions with some number of levels. These levels are not realized as *physical* dimensions: rather when a player enters a red square all the floor squares change color to match the level below the current one; a blue square will take the player up a level. Making sure such a maze is even solvable could be a substantial chore - the evolutionary algorithm could construct hundreds of such mazes, with certificates of solubility, per hour.

The fitness values obtained when fitness was maximized, rather than targeted to a particular number, were startlingly high. Since the fitness values are minimum number of moves required to safely traverse a puzzle, this leads to an undesired simplicity of the high-fitness puzzles; often there is only one choice of a way forward. Unless the player is not, initially, aware of what makes a safe move this means that the puzzles generated are rather simple and possibly tedious. This problem is solved in this study by the application of an alternate fitness function. Shifting to multi-criterion optimization that also rewarded branching factor in the implied maze or other difficulty or playability factors is an early priority for future research.

Dynamic programming is not a terribly “human” way of solving a maze. It is both exhaustive and memory intensive. This suggests that a maze that is hard for dynamic programming may not be hard for a human player and vice versa. Using humaniform agents in a second phase to grade the difficulty of evolved puzzles is another direction for future research on this project.

References

[1] A. Agapitos, J. Togelius, S. M. Lucas, J. Schmidhuber, and A Konstantinidis. Generating diverse opponents with multiobjective evolution. In *Proceedings of the 2008 Symposium on Computational Intelligence and Games*, pages 135–142. IEEE Press, Piscataway NJ, 2008.

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

[3] Bruce Albertson. *Chess Mazes*. Russell Enterprises, Milford, CT, 2008.

[4] 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.

[5] Daniel Ashlock. *Evolutionary Computation for Opimization and Modeling*. Springer, New York, 2006.

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

[7] C. Browne and F. Marie. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games*, 2:99–116, 2010.

[8] K. M. Bryden, D. A. Ashlock, S. Corns, and S. J. Willson. Graph based evolutionary algorithms. *IEEE Transaction on Evolutionary Computation*, 10:550–567, 2006.

[9] R. Hunicke and V. Chapman. Ai for dynamic difficulty adjustment in games. In *Proceedings of the 2004 AAAI Workshop on Challenges in Game Artificial Intelligence*, pages 91–96. AAAI, 2004.

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

[11] J. Marks and V. Hom. Automatic design of balanced board games. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE) 2007*, pages 25–30. AAAI, 2007.

[12] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[13] 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.

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

[15] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 111–118. AAAI, 2008.

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