Evolving Sparse Direction Maps for Maze Pathfinding

V. Scott Gordon
California State University, Sacramento
6000 J st., Sacramento, CA 95819
Email: gordonvs@ecs.csus.edu

Zach Matley
Digital Eclipse Software, Inc.
210 - 1965 West 4th Ave., Vancouver, BC V6J 1M8
Email: zachm@digitaleclipse.com

***Abstract*- A genetic algorithm is used to solve a class of maze pathfinding problems. In particular, we find a complete set of paths directing an agent from any position in the maze towards a single goal. To this end, we define a *sparse direction map*, wherein the maze is divided into sectors, each of which contains a direction indicator. Maps are evolved using a simple genetic algorithm. The fitness function samples the efficacy of the map from random starting points, thus estimating the likelihood that agents will find the goal. The framework was effective in evolving successful maps for three different mazes of varying size and complexity, resulting in interesting and lifelike agent behavior suitable for games, but not always the shortest paths.**

I. Background

Finding a path through a maze is a basic computer science problem that can take many forms. In this paper, we consider the case where a maze has a single goal location, and an agent must find a path to that goal from any arbitrary point. We start with a maze and a goal, and seek a solution which allows us to drop agents into random locations and have them quickly find the goal.

One approach to solving this problem would be to have each agent independently find a path from its current point to the goal, at the time that the agent is dropped, using any of a number of classical search techniques. Depending on processor speed, the resulting delay incurred when an agent is dropped might not be acceptable.

Another method would be to create a *complete direction map* for the goal. A direction map is a table of arrows identical to the maze, indicating a direction to move for each location on the grid. When an agent is dropped, it need only refer to the direction map to indicate which direction to move. A simple method for filling the direction map, based on Dijkstra's algorithm [1] is:

<u>Algorithm 1</u>

```
•  set global counter to 0.
•  set goal location in map to 0.
•  set other map locations to empty.
•  for each location with map number = counter
       for each adjacent empty location L
           place counter+1 in L
•  increment counter
•  repeat until no empty locations
```

Once the map is filled, an agent then moves simply by selecting an adjacent location which has a lower map number than that of its current location. An advantage of this method is that it only needs to be performed once, when the map is created.

A drawback to building a complete direction map is its size. Each instance of the map must contain a value for every location on the maze. This could consume more resources than necessary, depending on the type of platform and the number of copies of the map needed.

II. Sparse Direction Maps

Whereas a complete direction map contains an indicator corresponding to every location on a maze, a *sparse direction map* includes a single indicator for a block of locations. The maze is divided into equal-sized rectangular sectors, each of which contains a direction. The eight possible directions are in increments of 45 degrees. Figure 1 shows a 15x15 direction map for a simple 24x24 maze. Agents move in the direction specified by the map entry in which their actual maze location resides.

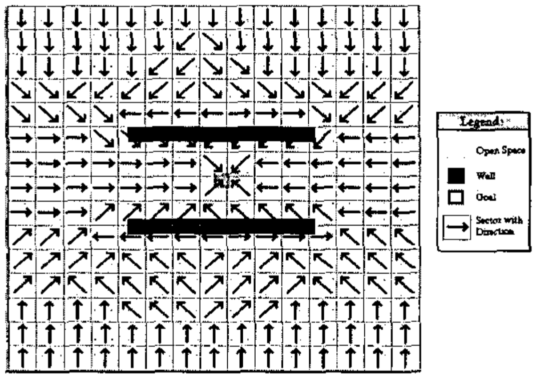

Figure 1. 15x15 sparse direction map for a simple 24x24 maze

Filling a sparse direction map using Algorithm 1 can lead to problems. Each sector contains multiple starting points, and in some cases a portion of the maze walls might cut across a sector, making it difficult to determine a single direction for the sector.

For example, consider the 9x9 maze shown in Figure 2. If the superimposed 3x3 sparse direction map is used, what single direction should be placed in the center sector? Although an upward arrow would be the optimal path for squares to the right of the barrier, it would lead to unrecoverable pitfall for agents dropped inside the arc of the barrier. Algorithm 1 is not guaranteed to produce a suitable direction for the center sector in this example.

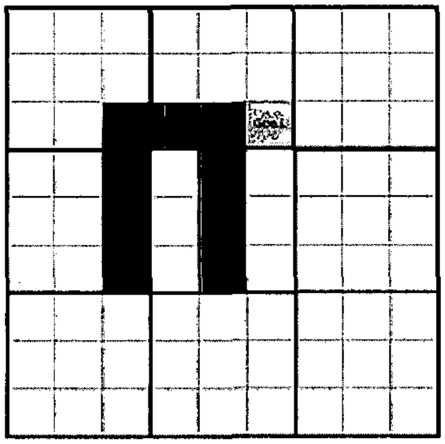

Figure 2. 9x9 complete direction map, showing 3x3 sparse sectors

Note that a sparse map represents a coarse-grained solution to a fine-grained problem, and that therefore the solutions found may not be optimal. For example, in Figure 2, although a downward arrow is the only feasible choice for the center square, it still results in a less-than-perfect map, because the cells immediately below the goal would also be set in the downward direction. The final solution would necessarily need to incorporate this compromise. Depending on the application, finding the very shortest path may not necessarily be important. In fact, indirect paths may be interesting in the context of games if they help facilitate lifelike behavior.

III. Genetic Algorithm Solution

Genetic Algorithms have been used previously for solving mazes. Previous approaches have focused on finding a solution between two points [2]. In contrast, our objective is to derive maps that direct agents from any starting point to a single goal. We populate our sparse direction maps using a Simple Genetic Algorithm (SGA) [3]. Genetic algorithms are search techniques based loosely on the concept of genetic evolution, where selection, crossover, and mutation are applied to a population of candidate solutions in order to evolve good solutions. Applying a genetic algorithm here requires an *encoding* and a *fitness function* specific to the maze problem. We also describe the genetic operators and parameters we used.

A. Encoding

Recall that a sparse direction map is a two-dimensional array of arrows, one per sector. Since there are eight possible directions, three bits can be used to represent them. Figure 3 shows the bit codes for each arrow. In a 15x15 direction map there are 275 directions of 3 bits each for a total of 675 bits per direction map. Thus a complete map can be encoded by concatenating all 275 3-bit sectors into a single 675-bit string. A population consists of several such strings, each representing a candidate direction map solution to a given maze. At the beginning, these bit strings are generated randomly, and the number of bit strings in the population remains constant.

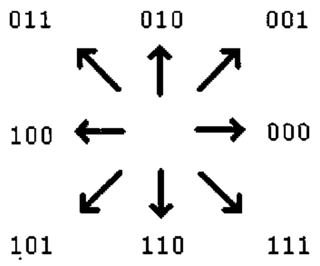

Figure 3. three-bit encoding for each map sector

This encoding has other applications. For example, it can be used to apply the genetic algorithm to the Knight's Tour problem [4]. Since the encoding is binary, standard SGA crossover and mutation operators can be used.

B. Fitness Measurement

The *fitness measurement* (or, *fitness function*) estimates how effective each candidate solution (string) is when used as a sparse map for a given maze. We wanted a fitness function that relied only on the success or failure rates of agents utilizing the map, and did not require any additional heuristic information or local searching.

The process we used for evaluating the fitness of direction maps is:

Algorithm 2 - fitness function

1) Launch a fixed number of agents from random positions in the maze,
2) Allow the agents to move according to their programmed behavior (which includes reading the candidate direction map) for a set number of steps,
3) The fitness measurement is the number of agents that reach the goal.

Fitness values therefore range from 0 to the number of agents launched. Note an advantage of algorithm 2 is that it requires no notion of distance to the goal. Choosing an appropriate number of agents to launch requires finding a balance between evolution time and accuracy -- running a large number of agents takes a long time. However, a small number of agents randomly distributed across the maze may provide a poor sample of all the possible starting points, producing less accurate estimates.

Since the fitness of a direction map is the number of agents that find the goal, it does not matter how each agent finds the goal. The genetic algorithm does not distinguish between short or long paths, and, as mentioned before, sometimes unexpected paths through the maze are generated. Also, the random factor of the fitness measurement means that repeated measurements on the same direction map will often yield different fitness values. This is because the starting points of the agents are randomly generated each time a map is evaluated.

C. Genetic Operators

We encoded three mazes: the small one shown previously in Figure 2, and two large ones described later in Section 5. The genetic operators we used varied slightly depending on which maze was being solved:

- **Selection:** 3-to-2 tournament. 3 strings selected at random, and the best 2 are selected.
- **Crossover:** standard 2-point with probability 100%
- **Population Size:** 200 for the small maze, and 100 for the other two cases.
- **Mutation Rate:** 0.5%
- **Agents used for fitness evaluation:** 250 for the small maze, 100 for the other two cases.
- **Maximum number of steps per agent:** 50

We also used a form of ***elitism*** at each generation. In an elitist genetic algorithm, the best fitness increases monotonically with each generation, typically achieved by ensuring that the individual with the highest fitness always survives to the next generation. Something similar was implemented for this project: once the fitness of each map is evaluated, a backup is made of the one with the highest measurement, which is then later copied into the next generation. This is similar to elitism, but since the fitness measurement is not constant, there is no guarantee that the best fitness in a population always increases.

A direction map is considered successful when *all* of the agents find the goal during fitness measurement. However, even when a map is successful, it is still possible that there are areas of the maze which were not tested, and from where agents would not reach the goal. For that reason, the problem is not considered solved until maps are found for which all the agents reach the goal in 3 successive generations.

IV. Agent

Agents, shown in Figure 4, are the characters which travel through the maze. Their responsibility is to move according to the directions stored in the map, avoid running into walls and recognize when they find the goal. In this project, agents travel the distance of a maze grid unit at each move. They can move in sixteen directions, even though the map only indicates eight. Utilizing sixteen directions helps in avoiding walls, and in making the agents' movement appear more lifelike by including a small amount of wandering.

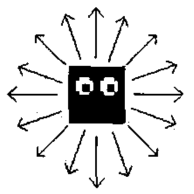

Figure 4. an agent

An agent usually moves in the map direction corresponding to its position in the maze. However, it must recognize when doing so would cause it to collide with a wall. When this happens, it alternately tries moving $d + a$ and $d - a$ degrees, where d is the direction from the map, and a starts at 0 degrees, and increases in increments of 22.5 degrees, until a safe direction is found.

Wandering involves augmenting the direction from the map by either 0, ±22.5, or ±45 degrees. The probability of wandering is adjustable - we used 20% for the large mazes and none for the small maze. Giving the agent the ability to wander helps make the agents' movement appear more lifelike, and may help speed up finding a solution.

V. Results

Figures 5, 6, and 7 show the direction maps evolved for a small maze (shown previously in Figure 2), a simple maze, and a complex maze. Solving the mazes required 28, 148, and 1220 generations respectively.

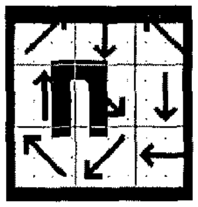

Figure 5. evolved direction map for the small maze (28 generations)

The maps as evolved, when used in conjunction with agent behavior as described in Section 4, produce successful yet lifelike maze traversal from every start point of each maze. Close examination of the sparse maps reveal that, in many cases, the paths to the goal are not always optimal. For example, in the small map (Figure 5), most paths lead around the barrier even when there is a more direct path. Further, at times the maps appear to lead the agent into a wall. Of course, the maps were evolved in the context of agent wandering and wall-avoidance, thus such directions can be acceptable.

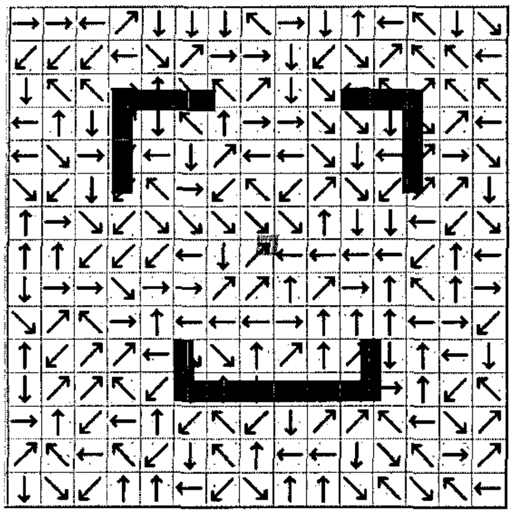

Figure 6. evolved sparse direction map for a simple maze (148 generations)

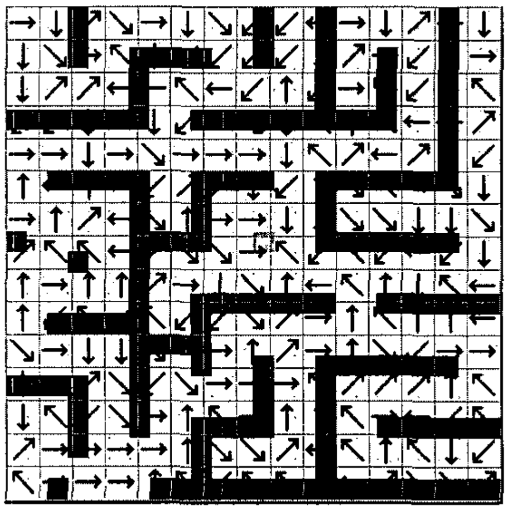

Figure 7. evolved sparse direction map for a complex maze (1220 generations)

VI. Conclusion

Previous genetic algorithm approaches to finding paths through a maze have focused on finding an optimal solution between two points. The objective of this project was to evolve data that would allow agents to quickly travel from any starting point to a single goal. Finding the shortest path length was not essential - successful but lifelike behavior was desired.

A data structure called a *sparse direction map* was proposed, which contains directions used to guide agents from any position to the goal. Maps were evolved using a variant of an elitist Simple Genetic Algorithm, incorporating a simple binary encoding and a fitness function that estimates the likelihood that agents will find the goal. This framework was able to evolve successful maps for three different mazes of varying size and complexity. The resulting maps led to interesting and lifelike agent behavior suitable for games, but not always the shortest paths. The Simple Genetic Algorithm proved to be an effective solution to our maze problem.

VII. Future Work

Although the SGA is popular for its simplicity and because it is well-understood, it is also known to be among the least effective as an optimizer [5]. Here too, while the SGA was able to find sparse direction maps for all of our test cases, evolution was rather slow. The complex maze required over 3 hours to solve on a Pentium 3. The experiments should be repeated with more effective genetic algorithms, such as Genitor, or one of the parallel models (Island, CGA, etc.).

The GA could be encouraged to find shorter paths by incorporating the total distance traversed by the agents into the fitness evaluation.

Further study needs to be done to determine how many agents should be used during fitness evaluation. A small number is faster and allows more maps to be considered, but a large number of agents results in more accurate fitness measurement. Perhaps utilizing a variable number of agents could balance the advantages of both.

Finally, direction maps could be extended to allow agents to chase a moving target. An often-touted advantage of genetic algorithms is that if a solution has been found, and the problem changes slightly, the algorithm can quickly adjust. *Dynamic* direction maps might then be an ideal case for utilizing the adaptive nature of genetic algorithms.

References

[1] Dijkstra, E., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik 1, pp. 269-271 (1959)

[2] Berg, A., "Genetic Algorithm - Maze Solver," (2003) http://www.sambee.co.th/MazeSolver/mazega.htm, 1996

[3] Goldberg, D., Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1975

[4] Gordon, V. and Slocum, T., "The Knight's Tour - Evolutionary vs. Depth-First Search," *2004 Congress on Evolutionary Computation,* Portland, Oregon.

[5] Gordon, V., Whitley, D., and Böhm, W., "Serial and Parallel Genetic Algorithms as Function Optimizers," *5th International Conference on Genetic Algorithms,* Urbana-Champaign, Illinois, pp. 229-235, 1993