Artificial Intelligence Solutions for Pathfinding for 2D Grids

*by Karl Ward*

# Introduction

I have been tasked with researching and implementing a 2D grid-based pathfinding solution in the form of an artificial neural network (ANN) or a genetic algorithm (GA). I have researched and compared the different solutions to this problem and have in turn implemented and tested the algorithm in a variety of situations. So far, I have implemented an A\* pathfinding algorithm which uses the Euclidean distance for the heuristic.

# Problem & Research

Melbourne Intelligent Computer Games has assigned the task of researching an ANN or GA that can provide an alternative to the deterministic A\* pathfinding implementation. During my research I have found a variety of peer-reviewed papers that consider the same grid style that

When researching this looked at ANN and GA implementations which maintain a focus on a 2D aligned grid and are both fast and practical for the purposes of a real-time game. First, I researched genetic algorithms and found 2 suitable papers on the subject and then found one suitable paper for neural networks. I have also found a paper detailing the use of ANN and GA for pathfinding in a more general sense.

The paper **Neural Networks for Real Time Pathfinding in Games** [1] presents a simple neural network solution that uses a simple 3-layer network. In this paper the authors acknowledge the need for real-time pathfinding in video games and they consider a changing world directly.

The advantages of this implementation are that it can avoid static and dynamic game objects by trying to give the neural network real-time awareness of the search-path’s surrounding cells. In this study the authors have already tested the effectiveness of the neural networking using various neuron counts on the hidden layer and have found that 4 neurons happen to be the fastest. This neural network’s weights are then evolved using a genetic algorithm.

The cons of this approach are that their implementation focus highly on a dynamic environment. Based on the task at hand this would be a lot of extra processing on features of the environment that don’t exist in our maps. Additionally, the authors conclusions have found that although this is possible using this approach it can become quite a slow process, and they propose a solution to be one ANN for travelling towards the target and another ANN for avoid obstacles, but this is out of the scope of the assignment.

In **Using a Genetic Algorithm to Explore A\*-like Pathfinding Algorithms** they approach the problem with a genetic algorithm. In this paper they use the idea of naval ships as obstacles within the grid, and they use a genetic algorithm to analyses around themselves and has a “natural” and “smooth” path for ships.

The advantage of this is a more natural path is found and smoothed as a result, it essentially tries to create a smooth path from start to end with speed and dynamic objects in mind. Also, in this example a larger map is used, which is much bigger than our own max map size of 20x20.

On the other hand, this paper requires a map which is quite open and mostly free of claustrophobic conditions, ones that would absolutely occur in our own maps & mazes. If the maps were smaller and had more obstacles it would then take long to solve the path.

**Evolving Sparse Direction Maps for Maze Pathfinding** [3] uses a genetic algorithm to create a path on a 2d grid with obstacles. This is achieved by segmenting the map into slightly large cells which group the actual cells and defines that section with a direction. The agents will then travel in the direction in which the parent cell is pointing towards.

The authors say that if you have a small not to complex map (ours is a maximum of 20x20), we can expect a result in a relatively short amount of time. This paper is also much more specific to requirements needs then the last paper as it considers a map in the same visual layout and the same degrees of freedom as our own. Additionally, with a maximum map of 20x20 it will be able to solve in a small amount of generations, however this may not be practical in real-time. A threaded implementation may be more practical.

On the other hand, the implementation noted that depending on the complexity of the map it can take up to a lot of time. This complex map of 15x15 took over 3 hours to solve on a Pentium 3. This is an older CPU and the implementation would fare much better on today’s CPUs like the one I’m testing on, but this would be nowhere near practical for a real-time pathfinding solution for games, especially for those games that run on a single thread. It states that it took 1220 cycles of the genetic algorithm to solve the 15x15 maze which may be feasible today if the algorithm was only run once at the beginning. Unfortunately, we need the implementation to be practical in a real-time process because the pathfinding algorithm may be called many times in a single frame by many different agents. Finally, they say that this will also not always find the shortest path, much like A\* the algorithm will stop once a path has been found, which may stop if the random generation are unlucky.

The paper **Pathfinding in Computer Games [4]** had a broad overview of many different deterministic and learning algorithms for pathfinding and explains the practical uses of ANN’s and GA’s. They detail many ways that ANN’s and GA’s can be implemented.

For ANN’s they are time efficient as it only uses multiplications and additions to run which are the “fastest operations” for a CPU to compute.

On the downside however, ANN’s cannot be to space efficient as you need many values for every neuron, bias and weight within the network. This returns to the space vs time problem that many programs face when dealing with large amounts of data.

Algorithm Choice

I have decided to implement an artificial neural network.

# Implementation and Evaluation

Before the test results I will mention the constants and variables I have used in my test. In my test rig I have used a 4th generation i7 4790K (4.00GHz). The tests will be done at non-overclocked base speed of 4.00GHz on a single thread. I also have 8.00GB of 2300MHz ram installed. The compiler I use is the Microsoft Visual C++ Compiler (SDK Version 10.0.17763.0). Using this compiler, I will provide a result for both debug mode and release mode to represent a development build and commercial build of the software.

The variable in these tests are the maps themselves in several forms. The first pairing is a small map of (10x10) and a large map (100x100) with natural obstacles that would be seen in a game. The next paring is a possible small (10x10) and large (100x100) but have the longest possible path to the solution for that grid size. Finally, I have a very large, possible and impossible map tests at a size of 1000x1000 (1,000,000 nodes). The last case unoptimized is generally unadvised but it should show that both algorithms can work on any sized map.

Mention

* + Memory Processed
  + Memory Needed
  + Time Taken in both Debug and Release Configurations

# Conclusion

* Describe the implementation process in the terms of a games company and how having used the paper has helped for their games development process.

# References

1. Graham, R., McCabe, H. and Sheridan, S., 2004. The ITB Journal: Neural Networks for Real-time Pathfinding in Computer Games. Volume 5, Issue 1, Article 21. Doi: 10.21427/D71Q95
2. Leigh, R., Shshil, J.L. and Miles, C., 2007. Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games: Using a Genetic Algorithm to Explore A\*-like Pathfinding Algorithms. Date 1-5 April 2007. Doi: 10.1109/CIG.2007.368081
3. Gordon, V.S. and Matley, Z., 2004. Proceedings of the 2004 Congress on Evolutionary Computation: Evolving Sparse Direction Maps for Maze Pathfinding. Volume 1, Date 19-23 June 2004. Doi: 10.1109/CEC.2004.1330947
4. Graham, R., McCabe, H. and Sheridan, S., 2003. The ITB Journal: Pathfinding in Computer Games. Volume 4, Issue 2, Article 6. Doi:10.21427/D7ZQ9J