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. More clearly stated in the first paper [1] the idea of an agent which moves step by step and has continuous amount of sensor data feed into the network allows the ability for the path to steer around objects.

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 using the implementation described in **Neural Networks for Real Time Pathfinding in Games** [1]. I will be using a Genetic Algorithm for the training and the Neural Network for calculating the path. With this implementation I will need to create a tailored fitness / cost function that will test how well a set of weights finds a path with the neural network.

I have decided to use this implementation because it uses a very small neural network (approx. 14 neurons) and the training method (GA) can be quite efficient on very large maps like the one presented in the paper itself. Neural Networks once trained can be quite efficient at producing the desired output as a NN performs simple matrix-vector multiplications and additions to achieve an output, and on such a small network the result

would be instantaneous. I am aware that using GA for my training method can yield a much longer training time, but the result can offer more varied results depending on the fitness/cost function I use to implement.

Implementation

During the implementation phase I ran into several pitfalls which have required a bit of out of the box thinking to achieve a reasonable result in a short enough time. The biggest problem I ran into happen to be the fitness / cost function. I cannot just increment the fitness when the cell moves because this would result in very high fitness values for a little to absolutely no change at all because there can potentially be a chromosome which will generate oscillations between 2 cells or maybe the chromosome walks back on itself to often. Both situations create high fitness value which then have a high chance of being selected for the next population which in turn results in a lot of generations which do not move or have very inefficient travelling paths.

I have implemented many different rules to make this function work effectively. Firstly, I have made it, so fitness is undone if the cell steps back on itself. This does mean that the neural network may be doing extra steps which are then discarded but this will at least give us a shorter path and not a path which goes forwards and back on itself. Another part to this function is something I brought into from the A* pathfinding algorithm and that is some sort of heuristic that alters the final fitness value. For example, if I have a path which on moves a square but the result is on the other side of the wall then this would seem to be very close to the end, but it still must travel around the wall increasing the cost like so:

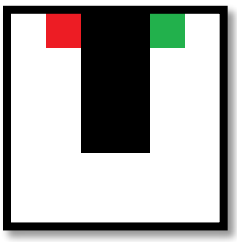


Figure 1: Example of a goal which is not far away heuristically but is far away on the path.

Therefore, paths which have not travelled very far have a small fitness which is then changed again by the heuristic of the current cell minus the target cell. This could defiantly be developed further but for now it will be able to find paths with sub-optimal performance / routing, which is more than suitable for my implementation. Finally, I cannot have the fitness function running indefinitely, so I have added a maximum amount of iterations (width * height * 0.5 + 1 (A path cannot possibly take more steps than this unless it is going back on itself.)) so it will stop so the genetic algorithm can keep evolving.

Another problem I found was the sizing of both the neural network and the “sensors” that have been described in the paper [1]. I have simply extended the sensors so that they stop when they hit any block at all and this had seemed to add extra complexity with very little payoff computationally, in fact none. Adding these extra inputs have made the whole process much longer which is what I have come to expect from my research. This is because I have used a genetic algorithm and from the research, they can take a long time.

In the genetic algorithm having a roulette wheel seemed to take a much longer time especially since it was a constant stream or random numbers. An alternative to this is the **Steady State Selection** [4], which selects only the fittest and generates a new population from those that are fittest. This does remove the randomness but due to the binary nature of the grid this will not have as much of an effect as say a 3D unaligned terrain map. With the number of rules on the fitness/cost function, it could produce a more effective result.

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. I have created 3 sizes of maps which are 5x5, 10x10 and 50x50, and for each size there is a natural map (for situations found in games) and a longest route. These are the map layouts (a red cell is the start and green cell is the end):

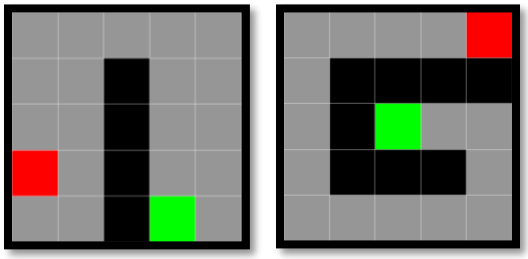


Figure 2: Small (5x5) Maps (Natural, Longest)

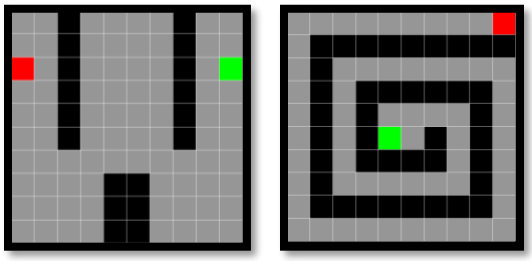


Figure 3: Medium (10x10) Maps (Natural, Longest)

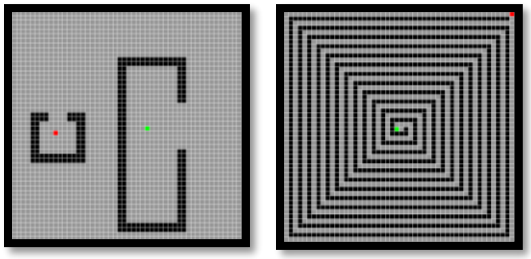
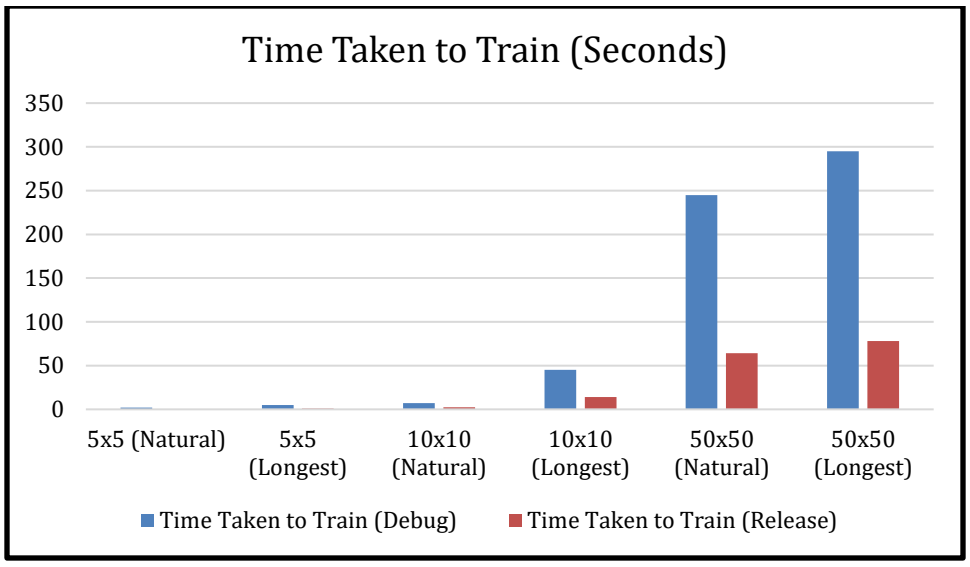
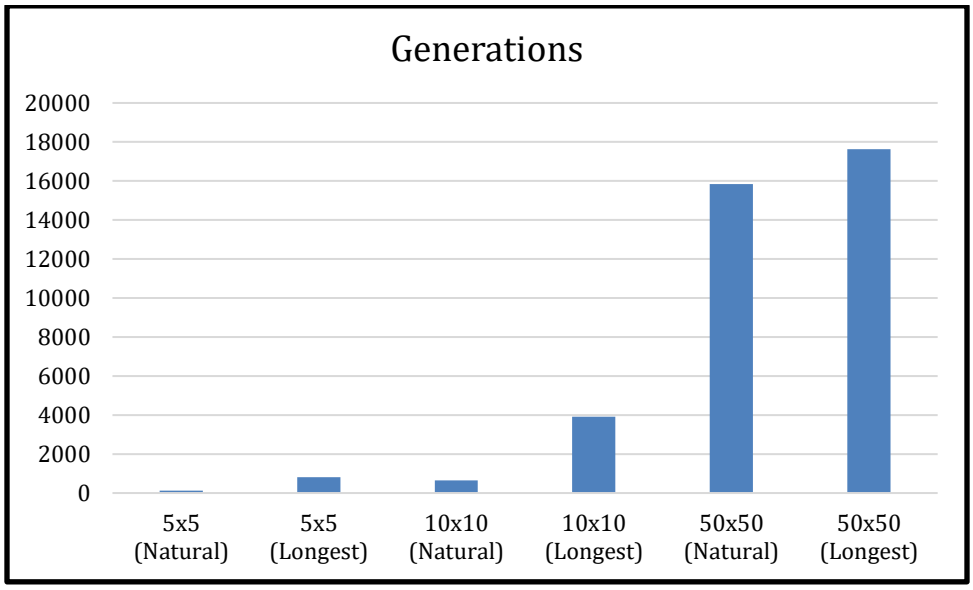
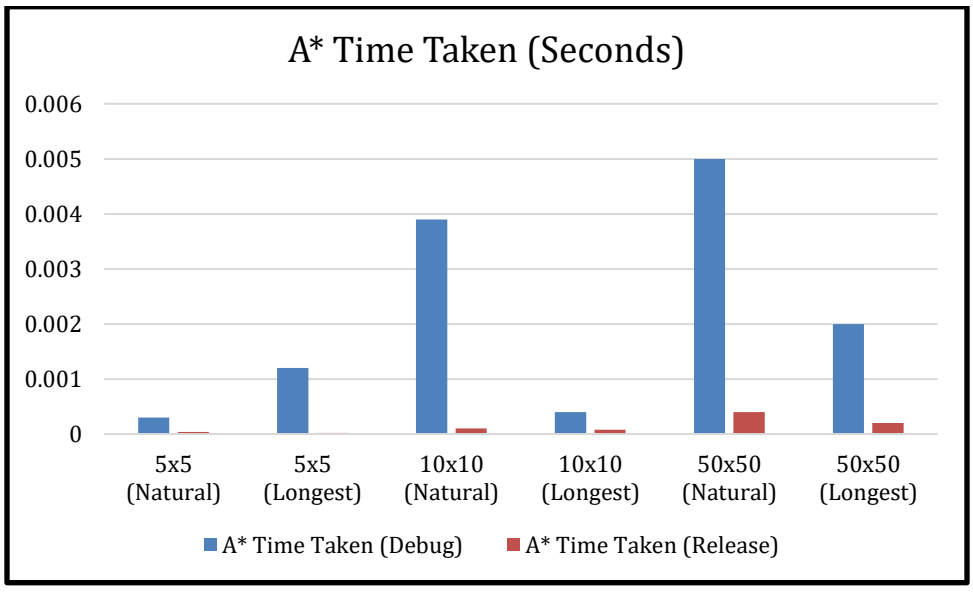


Figure 4: Large (50x50) Maps (Natural, Longest)

Here are the following training results:







From the results we can see that the artificial neural network is greatly underpowered when it comes to the training part. However, ones the weights are derived from the training on a specific map, they can be used to

find the path on a map very quickly as seen on the graphs. The ANN training is also greatly underpowered in debug mode which makes this not a very practical solution for development as ever-changing maps will need to be retrained constantly. Also, the training period gets exponentially larger as the map gets larger and more complex.

As you can see the A* algorithm is outperforming the ANN on many levels but between the 2 algorithms it seems that A* works best in narrow spaces, whilst the ANN works faster in open spaces with very few objects. The paths that are produced by the ANN is less then favorable, they are not particularly direct or efficient but with this being my first implementation I'm glad that it has managed to reach the end at all.

Conclusion

Implementing the Neural Network and Genetic Algorithm has been quite a task. Whilst training I had to wait several seconds or minutes (in some cases) for it to produce a path which can travel from the first node to the last node. This may work for static maps and then saving the training data for when that map is played, but for real time, no. I would have hoped it was a little faster at training, so a path would be found quickly. This is trained on the start and end nodes of one map, this would not be very good because as far as the neural network is concerned that is its entire world and only know that they are the start and end nodes. Training on more maps would allow for more possible paths but will also be more computationally expensive to train.

As mentioned in the implementation I ran into many different problems implementing a Neural Network and Genetic Algorithm for pathfinding, but I think this can be greatly improved by using back propagation instead of genetic algorithms and exposing all the map data much like image processing. Aided with the result of an A* path, the neural network could be trained using many maps with a close to optimum path. Allowing more information about the environment in which the neural network works I believe will greatly improve the performance and effectiveness of the path. This brings me to my next encounter.

When running my GA algorithm, I have noticed that It can take quite a long time to find the path and that path may not always be very clean. This can be a product of the path trying to dodge walls in a closed space, or it can be simply which it thinks a 'shortest path' would look like. This is not always the case however sometimes if the stars align the path produced will be very clean and sometimes even shorted then the path produces by the A* algorithm.

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