Optimal Path Finding in a GridWorld

CS7IS2 Project (2020)

Ashwin Sundareswaran R, Kavya Bhadre Gowda, Shubhanghi Kukreti, Chaudhary Guroosh Gabriel Singh

ramasuba@tcd.ie, bhadregk@tcd.ie, kukretis@tcd.ie, csingh@tcd.ie

Abstract. Many real life applications such as Intelligent Transportation System, Robot Navigation and Internet Routing can function efficiently when an optimal path between any two points is provided. Path finding algorithms are used to find optimal paths between two nodes in a graph like environment. In this project, various algorithms from different domains are evaluated and compared to find optimal path for traversal in a simulated grid world environment with obstacles.

Keywords: Path finding, Grid World, A-Star algorithm, Genetic algorithm, Reinforcement Learning.

1 Introduction

Optimal path finding can be implemented using various State-of-the-art algorithms like Breadth-First-Search (BFS), Depth-First-Search (DFS), A-Star (A*), Dijkstra's algorithm etc. In this paper we have implemented different classes of algorithm with different constraints. The first algorithm is A-Star which is an informed search algorithm. The second is Genetic algorithm (GA), which is based on meta-heuristics and improves the results based on the previous iterations. The third implementation is Q-learning and State Action Reward State Action (SARSA) algorithms which are part of Reinforcement Learning. These algorithms are based on an agent learning the environment by getting feedback on its actions.

The environment is a grid world which is defined by a size **M**, a start position, a destination and random obstacles. The grid dimensions can be unequal (MxN) but to get an idea of the grid size we have used square grids, where a grid of size M will have dimensions MxM with the start position on the top left and the destination on the bottom right. The obstacles can be either pre-defined or randomly placed such that there isn't an obvious direct path between the start and end positions. For the experiments we have used random obstacles with 20.5% obstacle density. Having low or high density gives straight forward answers, since there are usually straight paths with less obstacles and limited number of paths with more obstacles.

2 Related Work

A-Star algorithm
In [1] the A-star (A*) algorithm is defined along with the evaluation function *f(n) = g(n) + h(n)*. The admissibility and optimality of A* is discussed and proved, which is based on the heuristics function, h(n). Different heuristic functions are used and compared.

Genetic algorithm
In [2] the authors have tried to solve the shortest path problem with variable chromosomes length and genes by exchanging partial routes in cross over and selecting unique paths in mutation. This improved algorithm has solved the shortest path problem with a high convergence rate compared to state-of-the-art algorithms. In this the authors have specified the importance of population size selection, cross-over technique to get the diversified route and mutation to improve the local and global convergence.

Reinforcement learning
Reinforcement Learning(RL) is a form of Machine Learning in which the agent learns through observation of the environment. Based on its observations of the environment, the agent takes an action and receives positive or negative rewards. The main task of the agent is to maximize its rewards and find the optimal solution to reach a goal [3]. Q-learning is one of the simplest reinforcement learning algorithms which has been used to solve maze navigation problems and is an off-policy, model free algorithm. Using Q-learning, an agent is able to learn optimal control strategies through delayed rewards even though it has no prior information about the effect its actions have on the environment [4].

In [5] the authors have compared the use of different reinforcement learning techniques like Q-learning, SARSA in different applications and reported the comparison of the performance of these algorithms. It has been mentioned that the important part of the reinforcement learning is selecting the trade-off between exploration and exploitation techniques. SARSA is an on-policy learning technique. The agent estimates the next state and action value Q(s,a) by performing the action a, in state s, according to the formula as mentioned below[5]:
$Q(s,a) \to Q(s,a) + \alpha[r + \gamma * (Q(s+1, a+1)) - Q(s,a)]$

Where Q(s,a) is the current state and action. Q(s+1,a+1) is the next state and action. Reward is the obtained reward from performing the current action on the current state. Thus it uses the previous state and action to predict the next state and it's actions.

3 Problem Definition and Algorithm

The problem is to come up with approaches/algorithms to find the optimal path from start to destination in a grid world environment in, as little time as possible. A path p1 is better than another path p2, if the total length of p1 (from start to end) is less than the total length of p2 (from start to end). In reinforcement learning, the length of the path will also define the number of moves by the agent, i.e. "number of moves = length of path - 1".

3.1 A-star Algorithm

A-star (A*) algorithm was implemented to find the best path in any environment. The algorithm used is defined in [1] which traverses a graph, starting from the start node, until it finds the destination node. At any node, the algorithm looks at the non-visited child nodes and marks them as open, a node is further traversed only if a shorter route can be found by exploring that node. The algorithm uses an evaluation function which helps to traverse the graph optimally. The evaluation function at node n is $f(n) = g(n) + h(n)$, where g(n) is the distance of node n from the start node and h(n) is the heuristics between node n and the destination node. The Admissibility and Optimality of an A* algorithm depends on the heuristics function. If the heuristic is good, the algorithm always guarantees an optimal path.

For A* to work correctly and be efficient, the heuristics must provide as correct information as possible compared to the actual distance between the current node and the destination node. In general the heuristics should satisfy the following properties:

- The heuristic should not overestimate the actual distance.
- The heuristic should be as accurate as possible.

For the algorithm to be admissible and optimal, the 1^{st} point should be satisfied. For the algorithm to be efficient, the 2^{nd} point should be satisfied along with the 1^{st} point. The most optimal scenario is when the heuristic function returns the actual distance between the current node and the destination node. But this is not possible to evaluate since there might be obstacles in the environment. If the heuristic function always returns a 0 (or any other fixed constant), the algorithm proceeds forward based on the current nodes traversed. This is also called Dijkstra's algorithm, which becomes a special case of A* algorithm.

In our implementation there are 4 possible moves from any node, since no diagonal moves are allowed and therefore the shortest distance between the start and destination, without any obstacles, can be calculated using the Manhattan distance which is the optimal heuristics in our implementation.

3.2 Genetic Algorithm

Genetic algorithm (GA) is a self-adaptive search algorithm which is based on the idea of natural selection of population and genes. The genetic algorithm consists of population creation followed by iterations of crossover, mutation and population reduction.

Population Creation: The initial population creation is done by generating a list of **k** paths. Each path is a randomly generated path from the start node to the destination node. This is done using a random depth first search which returns the first occurrence of the destination. After this the population goes through iterations of crossover, mutation and evaluation until converge to a minima. The minima may be the global minima (can be verified by the A-star result) or can be a local minima. To decrease the chances of converging on a local minima, mutation is used.

Crossover: The crossover function is used to create new paths by taking paths from the existing population, known as the parents and creating new paths which is created by mixing sub-paths from 2 or more parents. We have implemented crossover by selecting different pairs of 2 parents and finding a random node R, occurring in both the parents. If the start and end nodes of the chosen paths are S1, S2 and E1, E2 respectively then the paths in the parents can be defined as $S1 \rightarrow R \rightarrow E1$ and $S2 \rightarrow R \rightarrow E2$, since both consist of a random node R. Therefore the generated paths after crossover will be $S1 \rightarrow R \rightarrow E2$ and $S2 \rightarrow R \rightarrow E1$.

Mutation: Mutation is used to avoid a local minima by introducing changes to a certain percentage of the population. This introduces the concept of exploration in the algorithm. In a path, a random node R is selected and a random path from either the start to R or from R to the destination node is found. So if the initial path is represented as $S \rightarrow R \rightarrow E$. After mutation, it could either be $S_{new} \rightarrow R \rightarrow E$ or $S \rightarrow R \rightarrow E_{new}$ with a 50% probability. This helps in preventing the algorithm from converging to a local minima.

Population Reduction: Since the population size increases after crossover, the total population count has to be restricted to a limit which may or may not be equal to the initial population count. Population reduction is done at the end of each iteration by using a fitness criteria. In our implementation the fitness criteria is the length of a path, lower length means a better path. Based on this, the population is sorted and top N paths are stored and the remainings are removed.

The iterations are stopped either when there is no significant improvement in the previous iterations or when a certain number of iterations are reached.

3.3 Q-Learning Algorithm

The Q-learning algorithm involves the interaction between an agent and an environment. The agent chooses an action A for a given state S which results in a reward R. When Q-learning is employed a Q-table or matrix is created which consist of values for State-Action pairs. At the beginning of the algorithm, the values in the Q table, called Q-values, are initialised as zero. But as the algorithm reiterates, the Q-values are updated after every episode. The agent uses the Q-table as a reference to decide the best action for a particular Q-value. For any given state $S_t \in S$, the agent must choose an action A_t from a set of available actions and change its state to S_{t+1} after receiving a reward r_t from the environment. The agent also learns the mapping between the states S and action A defined as policy . The agent has to follow the path that maximizes its total reward. If the agent hits an obstacle or does not follow the optimal path, it is penalised by getting a negative reward. The Q-values in the Q-table are updated using the following equation [3]:

$$Q(s_t, a_t) = Q(s_t, a_t)(1 - \propto_t (s_t, a_t)) + \propto_t (s_t, a_t) * [r_t + \mathit{\Gamma}_t max_a Q(s_{t+1}, a)]$$

where, $\propto_t$ is known as the learning rate which represents how much the algorithm prefers the old value v/s the new value. Learning rate value of 0 portrays that the agent does not learn anything whereas the value of 1 indicates that only the recently acquired knowledge will be considered by the agent and the knowledge gained in the past will be lost[3].

$\mathit{\Gamma}_t$ is known as the discount factor, determines the preference of the agent in choosing between the long-term high reward or the greedy reward; its value is between 0 and 1. Discount rates of one or more than one have proven to diverge the algorithm. As the value of $\mathit{\Gamma}$ approaches 1, the agent prefers to choose long-term reward [3].

r_t refers to the reward at time t. $max_a Q(s_{t+1}, a)]$ represents that the agent takes maximum of the future rewards and adds that value to the current reward.

These agents use exploration and exploitation strategies to find the next possible state. Exploitation is the way to find the next action derived on the obtained Q-value. This is an off-policy learning where the agent exploits the environment to find the path yielding the maximum reward. Exploration is the way of taking random action in the environment such that the agent tries to explore new paths to reach the destination. This helps the agent not to converge with local minima and gives way to explore new paths.This an on-policy learning method. It is always important for the agent to have trade-off between the exploitation and exploration values in order to learn the optimal solution.

3.4 State-Action-Reward-State-Action(SARSA) Algorithm

SARSA is an on-policy reinforcement learning algorithm. The agent learns from the action performed on the environment. It predicts the next action to take based on the action performed on the current state and the reward obtained. It is unlike Q-learning, where the agent looks for the maximum rewarded next-state. The agent interacts with the grid-world environment with grid size 10 * 10 and having random obstacles present between the starting point and the destination point. The agent receives a reward of -100 if it hits an obstacle in the way and a reward of +100 if it reaches the destination. The agent is trained for 1000 epochs and with the increase in the training time period, the agent learns the optimal path and follows the same to reach the destination. The formula for SARSA has number of hyper parameters which can be modified to tune the agent to learn accordingly. The following parameter value are selected.

Table 1. Hyper parameter value selection for SARSA agent.

Hyper parameter	Value
Epsilon	0.1
Learning Rate	0.1
Discount Rate	0.9

4 Experimental Results

4.1 A-Star (A*) Algorithm

Methodology: To see the effects of the heuristics on the efficiency and the optimality of the algorithm, we ran the algorithm across a fixed environment using different heuristic functions.

Table 2. Results of A* on a fixed 40x40 grid with random obstacles.

Heuristic function	Minima	Nodes visited	Nodes traversed
0 (a fixed constant)	81	1164	1164
Euclidean dist. $\sqrt{(X^2 + Y^2)}$	81	1086	1065
Manhattan dist. (X + Y)	81	625	559
$X^2 + Y^2$	85	170	106
2 * (X + Y)	85	210	141

Result and Discussion: Since diagonal movements are not allowed in our implementation, Manhattan distance between 2 nodes is an optimal and efficient heuristic function, since it is the closest we can get to the actual distance. Using under-estimated heuristic functions like Euclidean distance or a constant value also give an optimal result, but the efficiency is reduced and the algorithm takes longer to complete. On the other hand, using heuristic functions which overestimate the actual distance give non-optimal paths, although they may be designed to complete very fast. The results of different heuristic functions can be seen in Table 2.

Side Observation: Adding a constant value to the heuristic function does not change the behaviour. For example a function (X + Y + 4) will give the same results as (X + Y).

4.2 Genetic Algorithm

Genetic algorithms can be evaluated using 2 criteria:

- How accurate is the result to the global minima (in general: global optima).
- How fast the algorithm converges to a minima.

Test for algorithm correctness: To test the correctness, the algorithm was run multiple times using different mutation rates which is the percentage of the mutation population in each iteration. The code was run for various grid sizes and the local minima was compared to the global minima derived using A* on the same grid. The values of the minima obtained after 400 iterations are plotted (smoothed) for various grid size and mutation rate, can be seen in "Fig. 1". The algorithm was run 3 times for each grid size and the average was taken, therefore, for a 15x15 grid the algorithm was run on 3 different randomized grids of size 15.

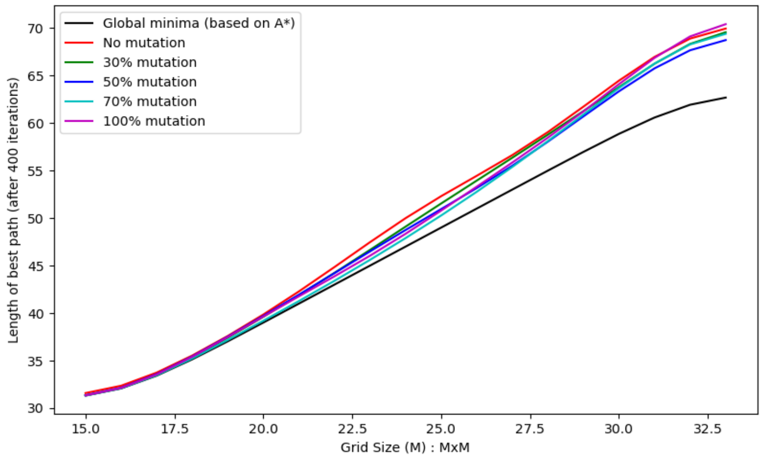

Fig. 1. Local minima for different mutation rates across different grid sizes.

Result: In Figure 1, it can be seen that for small environments (grid size less than 25), the best result is obtained when mutation rate is high (70%, 100%). As the environment size increases lower mutation rates (30% and 50%) give better results than higher mutation rates. Also, when the mutation rate is 0 i.e. no mutation, then the local minima is way off from the global minima, therefore some mutation is always required.

Discussion: It is clearly visible that mutation is necessary for the algorithm to explore and eventually find the global minima but finding the correct mutation amount can be tricky, specially for randomized environments. Even after finding an optimal mutation rate, the algorithm may not always converge to the global minima, especially for complex and big environments.

Test for Convergence: To test the convergence, we ran the algorithm through different sized environments and for each grid size the number of iterations were calculated in which the algorithm converges. In the experiment we say that the algorithm converges if there is no improvement in the previous 50 iterations. The algorithm was run 10 times for each grid size and the average was taken, therefore, for a 15x15 grid the algorithm was run on 10 different randomized grids of size 15. The mutation rate was fixed to 30% and the population size after every iteration was fixed to 50.

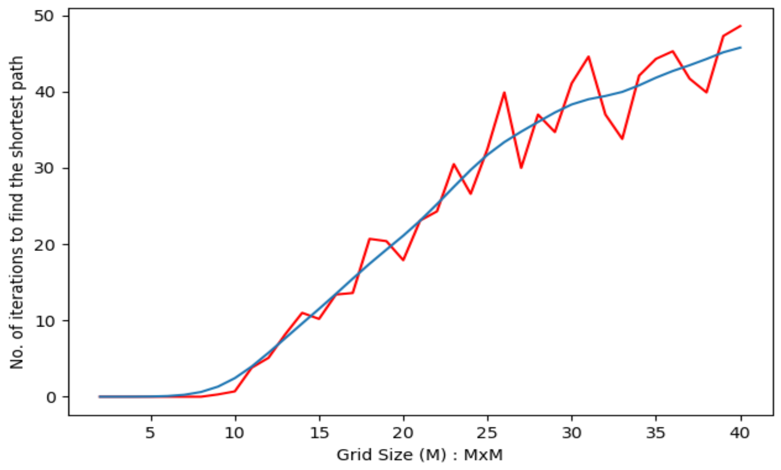

Fig. 2. Convergence rate of genetic algorithm across different grid sizes.

Results: In Figure 2, the Red coloured plot shows the actual values while the Blue coloured plot is the smoothed plot of the original values, which was implemented using a gaussian filter.

Discussion: As the size of the grid increases, the number of iterations required to reach convergence also increases. The increment in the graph is less than a linear increase, its because if the grid size increases the global minima will also increase and therefore the length of each individual in the population also increases. The algorithm works to find a longer path and therefore there is more scope for improvement after each iteration.

Genetic algorithms are also affected by other parameters like initial population count, population count after each iteration, amount of crossover, etc. Based on the requirement of the algorithm, the parameters can be fixed to improve the quality of the answer or be more efficient. Usually there is a tradeoff between the two, increasing the population size may help in the quality of the answer but decreases the efficiency.

4.3 RL Using Q-learning algorithm

Methodology: To test the Q-learning algorithm, random obstacles were introduced in the grid. The task of the agent was to avoid the obstacles and find an optimal path from starting point to the ending point. For each grid size, 1000 episodes were run with the parameters values being learning rate as 0.1 , discount factor as 0.9 and epsilon as 0.1. If the agent hits an obstacle, it is penalised by receiving a -100 reward. But if the agent reaches the target, it receives a reward of +100. After exploring the environment for a few episodes, the agent finally learns to navigate the grid without hitting any obstacle. Soon after that, the agent finds the optimal path and the algorithm converges and the final reward stays at 100. In "Fig. 3.", it can be observed that for a grid size of 10*10, the rewards are negative at first indicating the agent hitting an obstacle or straying away from the end point. But as the number of episodes increases, the agent ends with the same reward indicating the convergence of the algorithm.

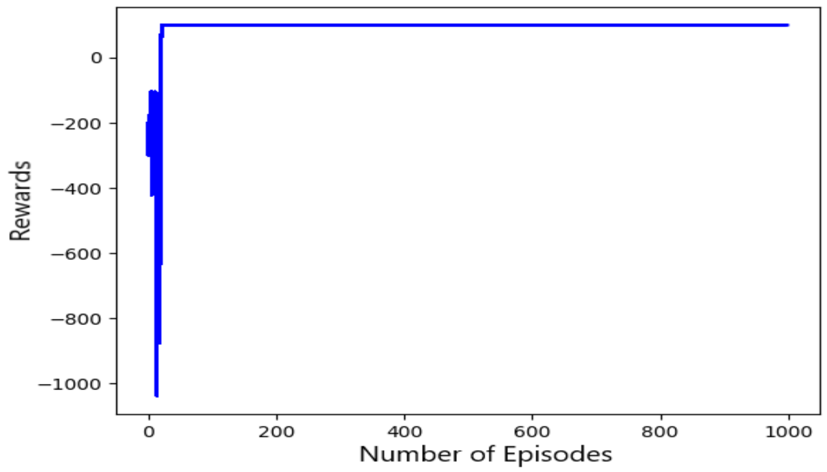

Fig. 3. Number of Episodes Vs Reward for Q-learning.

Additionally, the grid size was varied and the performance of the Q-learning agent was measured against the performance of A* algorithm by comparing the number of steps it takes for each algorithm to reach the target. Moreover, the time taken by the algorithm to converge was also calculated with the increase in the grid size.

Table 3. Time to Converge (in seconds) for A* and Q-learning algorithms.

Grid Size (MxM)	A* (seconds)	Q-Learning
5	0.43	1.75
7	0.99	2.29
9	1.07	2.58
11	1.89	3.03

Results and Discussion: The comparison in the performance of Q-learning agent and A*, our baseline algorithm, is shown in Table 4. The performance is measured based on the length of the path undertaken by an algorithm to reach the target point from the starting point. It can be observed that Q-learning provides similar path lengths as A* for small sized grids. But, as the size of the grid increases a difference between the optimal path lengths given by the algorithms can be seen.

Q-learning has been widely used in Reinforcement Learning applications, but the algorithm suffers from a drawback. It requires more time to reach the optimal solution. The time taken for the algorithms to reach the optimal solution is shown in Table 3.

Table 4. Final path length comparison of A*, Q-learning and SARSA algorithms.

Grid Size (MxM)	A* (Path length)	Q-Learning	SARSA
5	7	7	7
7	11	11	12
9	15	15	17
10	17	20	20
11	19	24	22

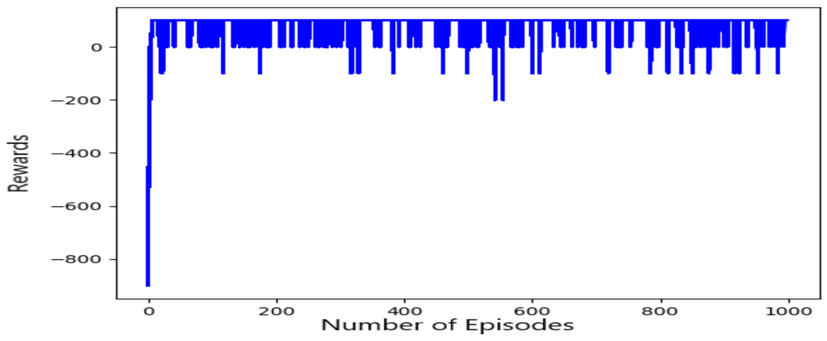

Fig. 4. Number of Episodes Vs Reward for SARSA Agent.

4.4 SARSA algorithm

The SARSA agent learns the environment and converges to the output after few iterations of training. “Fig. 4.“ shows the comparison between the number of training iterations and rewards. The agent gets more reward as it is getting trained. This implies that the agent learns the obstacle paths and then avoid hitting it once it knows the destination. It can be seen that after 20 iterations the agent learns the environment and achieves better rewards. The final results are shown in table 4.

5 Conclusions

A correct heuristics in A* always guarantees an optimal path. In other algorithms it was observed that a small number of iterations were required to find an optimal path when the grid size was small. But as the environment size increases, the algorithms require more number of iterations and more time to converge. It can also be concluded that genetic algorithm can be used in situations where a minima is required in a limited time but a global minima cannot be guaranteed. In RL algorithms, longer path lengths were returned by the algorithm when the grid size was increased.

References

1. P. E. Hart, N. J. Nilsson and B. Raphael, “A Formal Basis for the Heuristic Determination of Minimum Cost Paths,” in IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100-107, July 1968.
2. Chang Wook Ahn and R. S. Ramakrishna, “A genetic algorithm for shortest path routing problem and the sizing of populations,” in IEEE Transactions on Evolutionary Computation, vol. 6, no. 6, pp. 566-579, Dec. 2002.
3. D. Osmanković and S. Konjicija,“Implementation of Q — Learning algorithm for solving maze problem,” 2011 Proceedings of the 34th International Convention MIPRO, Opatija, 2011, pp. 1619-1622.
4. S. Manju and M. Punithavalli, ”An Analysis of Q-Learning Algorithms with Strategies of Reward Function”, International Journal on Computer Science and Engineering, vol. 3, no. 2, pp. 814-820, 2011.
5. Y. Wang, T. Li and C. Lin, “Backward Q-learning: The combination of Sarsa algorithm and Q-learning”, Engineering Applications of Artificial Intelligence, vol. 26, no. 9, pp. 2184-2193, 2013. Available: 10.1016/j.engappai.2013.06.016.