

Task 3. Reinforcement Learning

Garoe Dorta-Perez

CM50245: Computer Animation and Games II

March 18, 2015

1 Introduction

2 A* path finding

The A* algorithm can find paths optimally in a graph based on a heuristic. It works under the assumption that the distance heuristic will never overestimate the cost of the path and the path cost will not decrease as we travel through it. In a maze world, where the movement possibilities are north, south, east and west, with fixed positive costs. Using a Manhattan distance for the actual path cost and a euclidean distance to compute the heuristic will fit the assumptions.

In Algorithm 1 an overview of a general implementation of the A* in pseudocode is given. As stated in the previous paragraph, in our case, `graph.cost()` and `heuristic()` returns respectively, the Manhattan and the euclidean distance between two nodes. An example output with a small labyrinth where a path is successfully found is shown in Figure 1.

3 Q-learning in noughts and crosses

Q-learning is a reinforcement learning technique used when a problem can be modelled as a Markov decision process, where the world is modelled as a chain of *states*, where an agent can take *actions*, which is associated with a new state and a *reward*. The objective is to maximize the rewards that the agents gets, and by doing so taking the optimal actions in that given world. However, since the system is an online learner, there is a trade-off between *exploitation*, where the agent takes the actions that lead to the highest expected reward, and *exploration*, where the agent takes different actions to gain more information of unexplored states.

Each state s has a value $V(s)$ associated to it; the agent will choose either, a greedy *exploitation* action by performing an action a such that the next state s' will be $\max V(s_{t+1})$, or a *exploration* action a such that the next state s' will chosen randomly. When an end state is reached, the values for each state where a greedy action was taken will be updated by a backtracking mechanism as $V(s) = V(s) + \alpha [V(s') - V(s)]$, where α is the learning rate. In order to have a policy update which asymptotically guarantees converge, it is a necessary condition for α to decrease its value over time.

Algorithm 1: A*

Data: *goal* goal position, *start* start position, *graph* graph with the tiles in the map.

Result: *path* path from *start* to *goal*, *cost* total cost of the path.

frontier = PriorityQueue();

frontier.put(start, 0);

came_from = {};

cost_so_far = {};

came_from[start] = None;

cost_so_far[start] = 0;

while not frontier.empty() **do**

 current = frontier.get();

if current == goal **then**

 | break;

end

for next in graph.neighbors(current) **do**

 new_cost = cost_so_far[current] + graph.cost(current, next);

if next not in cost_so_far or new_cost < cost_so_far[next] **then**

 cost_so_far[next] = new_cost;

 priority = new_cost + heuristic(goal, next);

 frontier.put(next, priority);

 came_from[next] = current;

end

end

end

path = getPath(came_from);

goal = cost_so_far[current];

```

X 00000000001111111111
01234567890123456789
Y
00 S*****
01 .#####*
02 .#..###.#.#.###*
03 .##..###.#.#.###*
04 .#...##.#.###*
05 .#..#...***.***.
06 .###.....*###.....
07 .#####.*.#####.
08 .#.....G#.....
09 .#####.#####.
10 .#...#.#.....
11 .####.#.#.#####.
12 .#.#.###.#.#.###..
13 .#.#.###.#.#.###..
14 .#...##.#.#...##..
15 .#..#...#.#.....
16 .###.....###.....
17 .#####...###.###
18 .#.....#.....
19 .....

```

Figure 1: Path finding example, S is the start position, G is the goal position, $*$ are the path points, $.$ are empty spaces and $\#$ are walls.

```

---- Policy ----
X = Hit, Blank = Stick

With usable ace | Player hand value
                | 21
                | 20
                | 19
                | 18
X X X X X X X X | 17
X X X X X X X X | 16
X X X X X X X X | 15
X X X X X X X X | 14
X X X X X X X X | 13
X X X X X X X X | 12
X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

Without usable ace | Player hand value
                  | 21
                  | 20
                  | 19
                  | 18
X                  | 17
X                  | 16
X X X X           | 15
X X X X           | 14
X X X X           | 13
X X X X X X X X | 12
X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

```

Figure 2: Blackjack policy example, with 20% exploration rate after 100000 games.

```

---- Policy ----
X = Hit, Blank = Stick

With usable ace | Player hand value
                | 21
                | 20
                | 19
                | 18
X X X X X X X X | 17
X X X X X X X X | 16
X X X X X X X X | 15
X X X X X X X X | 14
X X X X X X X X | 13
X X X X X X X X | 12
X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

Without usable ace | Player hand value
                  | 21
                  | 20
                  | 19
                  | 18
X                  | 17
X                  | 16
X X X X           | 15
X X X X           | 14
X X X X           | 13
X X X X X X X X | 12
X X X X X X X X | 11
1 2 3 4 5 6 7 8 9 10 -> Dealer seen card

```

Figure 3: Blackjack policy example, with 40% exploration rate after 100000 games.

Nought and crosses can be modelled as a Markov decision process given that a board state characterizes the sum of the moves that lead to such state. The *agent* is one of the players, a *state* is a given configuration of a board and the *reward* is given when a game ends. The agent will receive a high positive reward when it wins, a negative reward when it loses and a small positive reward when drawing.

In the implementation the state space will be modelled in a tree as shown in Figure 4. On the root of the tree, we have empty board, the children are the 9 possible plays for the crosses player, who will always be the first to play.

To test our system, the agent would play a fixed number of games against an AI with fixed moves initialized at random. In Figures 5, 6, 7 and 8 the behaviour of the system under variations of the exploration rate are shown.

4 Q-learning in BlackJack

5 Results and conclusions

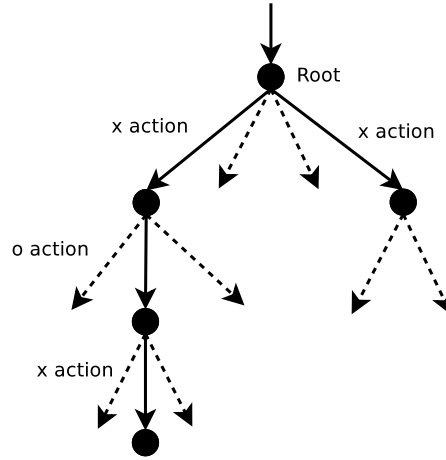


Figure 4: Noughts and crosses tree state.

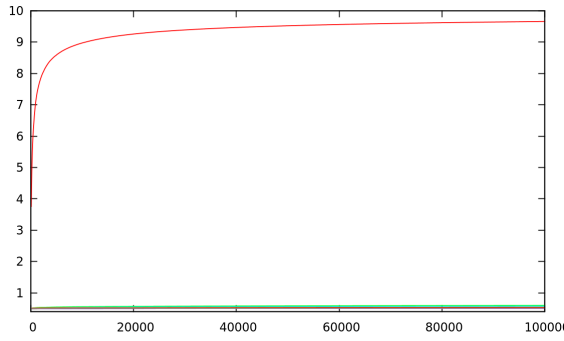


Figure 5: Values for the first movement for noughts and crosses with 20% exploration rate after 100000 games.

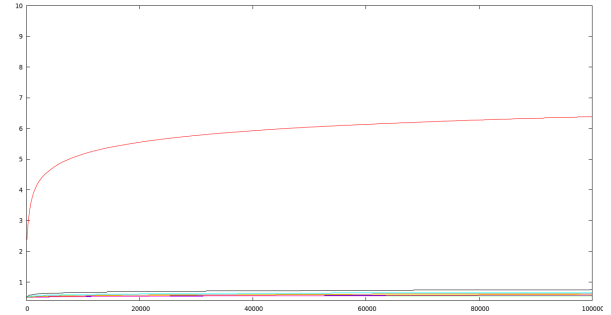


Figure 6: Values for the first movement for noughts and crosses with 40% exploration rate after 100000 games.

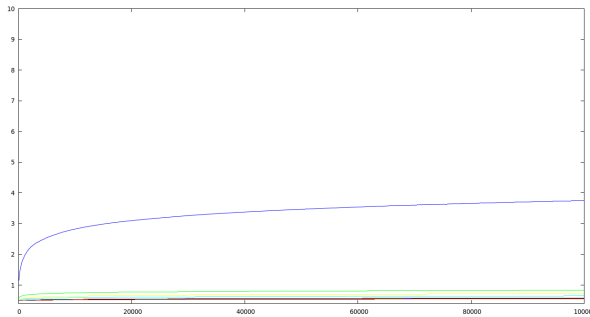


Figure 7: Values for the first movement for noughts and crosses with 60% exploration rate after 100000 games.

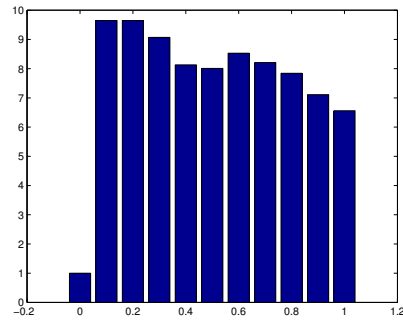


Figure 8: Mean gain after playing 100000 games of for noughts and crosses with the given exploration rates.