Student ID: 1681669

Course: MSc Computational Neuroscience and Cognitive Robotics

Assessment Title: Combining Markov Decision Processes and Logical Reasoning for Path planning

Practical Tutor: Dr. Mohan Sridharan

Word Count: 2926

Combining Markov Decision Processes and Logical Reasoning for Path planning

Abstract— **Markov Decision Processes are methods that can be used to attempt to optimise expected reward for any given state of an agent. Markov Decision Processes can become computationally expensive in large domains, such as during path planning, as many possible states exist. Decreasing the domains size decreases computation time but also decrease accuracy of the route planned. Logical reasoning could be used to limit areas needing to be pathed through using a Markov Decision Process, and as such limit the size of the domain to only necessary regions. Answer Set Programming is one method for logical reasoning that can be used, and this paper utilises it for pathing through a map split into rooms. Optimal values for the Markov Decision Process for agents with different movement accuracies were also found. Answer Set Programming was suitable and able to be used to plan a route through different rooms in the map to reach a goal. Markov Decision Processes were suitable for high movement accuracies, however for low accuracy movement went too close to 'walls' for the movement to be considered safe. The Markov Decision Process likely failed for low accuracy movement as the method used for planning in advance assumes the agent is always in the expected state, which is highly unlikely for low accuracy movement.**

I. Introduction

Path-planning is a well-studied area within robotics, as planning optimal routes which ensure both speed and safety is important for many tasks. Methods such as: A[1], D* [2], and D* Lite [3] work by minimising 'cost' (time taken to reach the goal, or distance from wall) to create an effective route. Markov Decision Processes can also be utilised for path planning (MDP) [4]. These work by calculating expected reward from any 'state' an agent could be in. MDPs use value iteration, which update previous expected rewards to create an optimal plan. The agent then moves from state-to-state to increase the expected reward of its state. MDPs can be computationally expensive in large domains, because as map size increases, so do the number of calculations required.

Decreasing map-resolution, decreases the accuracy of the path planned, but does decrease computation time. By limiting the MDPs formed to smaller areas, and only pathing through relevant areas of the map, the processing time would be decreased as each MDP calculated would contain fewer potential states an agent could end up in.

Splitting maps into sections and using reasoning to determine which sections to path through could enhance efficiency . Answer Set Programming (ASP) is one method employed to perform logical reasoning [5]. This involves creating a knowledge base and a domain with rules. ASP can determine what is 'true' in the domain from these rules. Planning is done by defining actions that can occur at a given time-step. By limiting time-steps, minimal plans are formed.

The present study aims to merge MDPs and ASP to create an effective pathing method. This is achieved by splitting the domain into 'rooms', forming an MDP for possible goal-states in each room, then using ASP to determine which rooms must be pathed through to reach a goal-state (in this case, doorway into another room, or the final goal position).

It is hypothesised that by combining ASP and MDPs an effective pathing technique will be created, and that by combining different 'rewards' for the MDP (hit-wall and reached-goal) a suitable path will be planned for an agent. It is also hypothesised that for path planning an MDP will be more effective for an agent with more accurate movement than less accurate movement.

II. Answer set programming

Answer set prolog (ASPr) is a programming language that enables knowledge representation and logical reasoning. ASPr has been used in robotics to enable logical reasoning [6,7,8]. The version of ASPr used in this research was compatible with the SPARC system [9]. SPARC requires ASP to be split into sorts, predicates, and rules. The Sorts section defines the variables (location, robot, and step), the fluents (inertial or defined), and the actions available within the domain. Locations are rooms in the map, and steps are the total number of actions allowed to be taken. The only action in the domain is move, written as:

move(robot,location(x)).

This states that the robot has moved to location x. Inertial fluents can be altered by actions, such as the inertial fluent 'at', which represents the agent's current location on the map. Actions cannot alter defined fluents. The only defined fluent in this domain is adjacent, defining how rooms are connected. This is written as:

adjacent(location(x), location(y)).

Predicates define relations within the domain. Predicates used are: holds(fluent,step), occurs(action,step), success(), goal(step), something_happened(step). The 'holds' predicate defines that a fluent 'is true at a given time step. The same principle applies for occurs but is for actions instead. Goal and success are used to define when the goal-state is reached, and something_happened ensures actions occur at every possible time-step, unless a goal-state is reached.

Rules define what can occur in the domain. To define that a robot cannot be in two places at once the following rule is used:

```
-holds(at(R,L2),I) :- holds(at(R,L1),I), L1 != L2.
```

This rule states that R cannot be at L2 at time I so long as R is at L1 at time I, and that L1 does not equal L2. Furthermore, to define that the robot can only move to adjacent rooms the following rule is used:

```
holds(at(R,L),I+1) :- occurs(move(R,L),I),  holds(at(R,LP),I),
                      holds(adjacent(L,LP),0),      LP != L.
```

This states that R is at L at time I+1 if R moves to L at time I, R was at LP at time I, L is adjacent to LP at time 0 (functionally equivalent to time I as it is assumed defined fluents do not change) and that LP is not equal to L.

III. METHOD

The program written has four main parts: running the experiment; the MDP; reading an inputted map; and integration with ASPr. The program takes as input a map, a start co-ordinate, an end co-ordinate and the reward scheme. As output the robot provides either a route through each room in the map from the start to the goal coordinate, or an error if no route was able to be found. Each room is processed individually so it is possible for a route to be only through relevant rooms. The entire codebase can be found here: https://github.com/jandfranc/ASP-Robot-Project

The map reading function reads an input black and white map with obstacles removed and splits it into individual rooms which are assigned a number. 'Doors' (holes in the wall of the room) are identified and used to assess how rooms are connected to other rooms. Rooms were defined as rectangular white space surrounded by black and were identified by checking the top left corner and bottom right corner to assess if these regions fit to a given pattern. Doors were defined as gaps on the wall of a room and were able to be identified so long as the door was not directly lined up with a vertical edge of the room. Rooms and doors could be of any size. The map utilised can be seen in Figure 1, as well as the start and goal point used for assessing the MDP.

An MDP is used to create plans for each room of the map. To calculate a plan, the MDP takes as input an array corresponding to the room, reward values for wall collision, reaching the goal state (RG), and movement accuracy(P_r). The virtual agent has a probability equal to P_r of moving in the intended direction, and a probability of:

```
(1-movement_probability)/2
```

of going to either of the adjacent tiles instead. Expected reward is calculated by finding the reward (or expected reward from previous iterations) in each possible direction (up, down, left, or right) by multiplying the reward from the intended spot by P_r and the expected reward of the two adjacent tiles by P_w. The reward for the action with the highest expected reward is set as the tiles expected reward. Movement can only occur on non-goal state tiles (in the case of this experiment, there is only one goal-state per MDP. If the point is the goal point, the tiles value is set as RG. This runs iteratively, using the previous values as the baseline values for the next iteration. This is known as 'Value Iteration'. To ensure a successful plan is formed, discounting is used. Discounting is explained here:

$$O = V \times D^i$$

Where O is the outputted value for a tile, V is the highest possible reward for a tile, D is the discount, and i is the number of iterations. Discounts less than 1 mean later values are less utilised than earlier values. This program checks to see if the current V is the same as any of the previous values for the tile, and if not changes it to O.

For path planning, the agent's initial position is set in the first room. For all other required rooms it is set as the centre point of the door which was used to enter the room. From this initial point all surrounding tiles are checked and the tile with the highest expected reward is chosen to be the next tile to move to. This process until the current state is the goal state, at which point the route is complete, and the route being calculated is switched to the next room. It is possible the plan computed will not route to the goal. To determine if this is the case the route planning function checks whether the next coordinate has previously been in the route. If this is true, the route is classed to have failed, skipping the current room and switching to path the next required room.

To decide which of the rooms need to be routed through, ASP is utilised. A template ASP file is used, and rules are added to it based on the connections of the room. These rules are of the form:

```
holds(adjacent(x1,x2),0).
```

This rule represents that room x1 and x2 are adjacent and this holds true at time step 0. For the purpose of this experiment, rooms adjacent at time step 0 are assumed to always be adjacent (i.e. doors between rooms cannot be closed). These adjacency rules are created automatically for each connection on the map. Importantly, rules must be available for adjacency between x1 and x2, and adjacency between x2 and x1, otherwise movement can only happen in the direction specified. If only one of these is written, then movement will only be able to occur from x1 to x2, and not from x2 to x1. Two other rules are added to the ASP file, which are the start room and the goal room. These are of the form:

```
holds(at(r,x1),0).
```

and

```
goal(I) :- holds(at(r,x2),I).
```

The first rule means robot is at x1 at time 0. The second rule means the goal is achieved at time I, if robot is at x2 at time I. The ASP file is started to try and solve the optimal route. To optimise solving times, a maximum number

of timesteps allowed is used. This starts at 1 and increases by 1 until a solution is found. If no solution is found after this number is larger than the number of rooms in the map, it is assumed no route is possible. Assuming a route is found, the robot then uses the path planning for each room described above to determine a route to the final goal state.

To run the experiment, required inputs are a map, a start co-ordinate, a goal co-ordinate, rewards for each action (as described above) and number of iterations for the MDP. For performing the experiment, the independent variables used will be the possible reward values, the discount, and the movement probability. The 'hit wall' reward value will be set as -100, -10, or 0. The 'goal' reward value will be set as 10, 100, or 1000, as demonstrated in Figure 1. Movement probability will be set as 0.3, 0.8, and 1. Discount will be set as 0.5, 0.9, and 1.

To determine if logical reasoning is suitable for path planning, it will be assessed to see if ASP can plan a minimal route. This is determined by whether the route taken goes through as few rooms as possible. A path in both directions from 0 to 5, and 5 to 0 will be assessed, and a minimal path would not pass through rooms 2 and 3, as this requires more rooms to be pathed through.

The dependent variable used will be the total length of the route planned from room 0 to room 5 for each combination of independent variables, as well as whether a route has been found. Furthermore, for Pr of 0.3 or 0.8, the 'safest' (not close to wall) route that is minimal will be chosen. Routes that are not minimal, or 'safe' will be excluded.

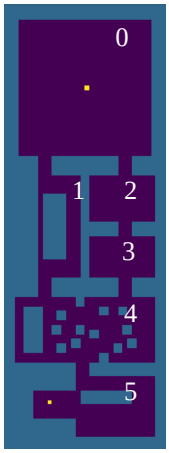

Figure 1. The upper yellow dot represents the start point. The lower yellow dot represents the goal point. The map here is the map used for the experiment. The numbers in the top left represent the room number.

IV. Results

The ASP program was able to suitably path through the room, when pathing in either direction as the plan did not pass through rooms 2 and 3 under either circumstance.

All routes planned when discount was set to 1 failed, with no route found for any room on the map. All routes planned with a discount of 1 were deemed ineffective and as such were removed from further analysis. As a discount of 1 means no values were discounted, eventually all values became similar. An example of an MDP created from a discount of 1 can be seen below. Due to the lack of discounting it is not possible for a suitable route to be planned. The minimal route here was 89 points long, which is the minimal possible route.

For a P_r of 1 the optimal route was the shortest route. All possible plans (excluding for a discount of 1) gave the exact same route, regardless of the parameters set. This is unsurprising as the hit-wall probability has no impact as there is no chance for wall collision, and goal reward makes no difference so long as it is greater than 0. The map planned can be seen in Figure 2.

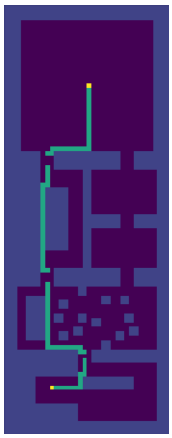

Figure 2. The optimal route planned for a Pr of 1. The yellow areas are the start and end points, and green points are the route planned.

For a P_r of 0.8, routes which did not avoid small gaps (i.e. one tile wide gaps) were excluded, as well as routes which did take a minimal path but were closer to walls than other minimal routes. Minimal paths occurred when discount was at 0.5, hit-wall was any value except 0, and the reached goal value was either the same or greater than the hit wall value. There were 5 minimal plans, and the routes were 129 points long. The route can be seen in figure 3.

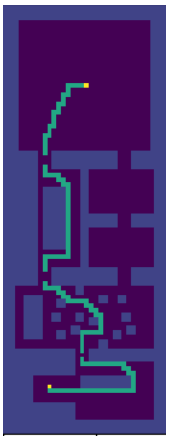

Figure 3. The optimal route planned for a Pr of 0.8. The yellow areas are the start and end points, and green points are the route planned.

For a P_r of 0.3, routes which did not avoid larger gaps (i.e. two tile wide gaps) were excluded. Furthermore, routes which took larger gaps, and were further from walls were considered more optimal. Paths of minimal length but took a path closer to the wall were removed also. The only path removed due to this criterion was for a discount 0f 0.5, hit-wall of -100, and reached goal of 100. The optimal routes were found when discount was set to 0.5, hit-wall punishment was -10, and reached goal was 10, and 100. The other plan this occurred for was when discount was 0.5, hit-wall was -100 and reached goal was 10 or 1000. The routes planned despite the low movement probabilities were still close to the wall in most instances. There were 4 minimal plans, and the routes were 131 points long. The route can be seen in figure 4.

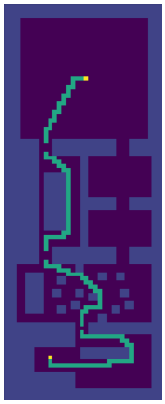

Figure 4. The optimal route planned for a Pr of 0.3. The yellow areas are the start and end points, and green points are the route planned.

V. Conclusion

This paper aimed to assess how well a combination of logical reasoning and planning can be utilised to optimise an MDP for the purpose of planning a route through a map split into rooms. MDPs were formed for different movement probabilities to resemble robots with different movement accuracies. ASP was effective at planning as it created a minimal route through a given domain. Furthermore, use of an MDP for path planning was effective for high accuracy movement, but not for low accuracy movement.

For perfect movement ($P_r = 1$) the path planned was optimal and took the minimal route possible, thus showing that the MDP is suitable for path planning. This in combination with ASP was suitable for planning paths through multiple rooms. For a P_r of 0.8 the path planned took a different route due to imperfect movement, as can be seen by not going through the thin gap in room 1, and on room 4 not going between the small gaps and instead taking a longer route. For a Pr of 0.3 the paths planned were not good and were often similar to 0.8, as the 0.3 route is only 2 points longer than the 0.8 route. This is insufficient as for a Pr of 0.3 it is important to not be close to walls to avoid collision.

The issue with paths planned at a Pr of 0.3 is likely due to how MDPs work. MDPs give an 'optimal' route to maximise reward, so when probabilistic movement moves the robot to an unexpected location, it can correct based on the MDP. This means for perfect, or high, movement probabilities path planning works well, as the new position is (or is almost always) the expected position also. For low Pr however, it is unlikely to end in the correct position, so the advantage of an MDP in correction from an unexpected state change is not utilised. As such, pre-planning a path for a robot with low Pr is likely not suitable, and online movement choice is likely more useful. This can still be combined with logical reasoning however, to allow for a general plan to be determined, as this would still allow smaller MDPs to be utilised.

This paper has shown that ASP and MDPs can be integrated to form effective path finding methods, so long as the agent is mostly accurate in its movements. ASP could also be combined with methods other than MDPs, such as A* and D* to enhance their efficacy and processing time by limiting planning to relevant domains. Furthermore, this principal can be applied to other decision-making methods, such as speech processing, as by using known knowledge to determine context, processing could be limited to only relevant domains.

References

[1]
N. J. Nilsson, *Problem-solving methods in artificial intelligence*. New York Etc.: Mcgraw-Hill, 1974.

[2]
D. Ferguson and A. Stentz, "The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments," 2005.

[3]
S. Koenig and M. Likhachev, "D* Lite," *In Eighteenth national conference on Artificial intelligence*, 2002.

[4]
S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: Mit Press, 2010, pp. 487–509.

[5]
P. Cabalar, D. Pearce, and A. Valverde, "Answer Set Programming from a Logical Point of View," *KI - Künstliche Intelligenz*, vol. 32, no. 2–3, pp. 109–118, Jun. 2018.

[6]
E. Aker, V. Patoglu, and E. Erdem, "Answer Set Programming for Reasoning with Semantic Knowledge in Collaborative Housekeeping Robotics," *IFAC Proceedings Volumes*, vol. 45, no. 22, pp. 77–83, 2012.

[7]
T. Mota and M. Sridharan, "Commonsense Reasoning and Knowledge Acquisition to Guide Deep Learning on Robots," Robotics: Science and Systems XV, Jun. 2019.

[8]
M. Sridharan, M. Gelfond, S. Zhang, and J. Wyatt, "REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics," *Journal of Artificial Intelligence Research*, vol. 65, pp. 87–180, Jun. 2019.

[9]
iensen, "iensen/sparc," *GitHub*, 26-Dec-2019. [Online]. Available: https://github.com/iensen/sparc. [Accessed: 01-Jan-2020]