Introduction

The task of path planning is a well-studied and important area in the field of intelligent robotics as optimising a route so it is safe and quick is important for many tasks. There are a variety of methods utilised, such as A\* (Nilsson, 1974), D\* (Ferguson & Stentz, 2005), and D\* Lite (Koenig & Likhachev, 2002), which work by minimising ‘cost’ (time taken to reach the goal, or distance from wall) to create an effective route Another method utilised for path planning is Markov Decision Processes (MDP). These work by calculating expected reward from any ‘state’ (or position, in this case). MDPs can be computationally expensive, especially in large domains which is often necessary for path planning. Furthermore, using value iteration these calculations are run incrementally many times to achieve a close-to-optimal plan. As map size increases, calculations required increase exponentially as, for a square domain:

INSERT EQUATION HERE

A trade-off is to decrease the resolution of the made but this decreases the accuracy of the map-planned. By limiting the MDPs formed to smaller areas, and then only pathing through relevant areas of the map

By splitting the map into parts and then using logical reasoning to determine which parts must be pathed to a better solution could be achieved. Answer set programming (ASP) is one method to perform logical reasoning. This involves creating a knowledge base, and a domain with rules. ASP can then determine what is true in the domain from the rules set. Planning can be done by defining actions that can be performed at a given time-step. By limiting available time-steps, minimal plans can be achieved.

This study aims to merge MDPs and ASP to create an effective path planning method. By splitting the domain into ‘rooms’, forming an MDP for possible goal-states in each room, and then using ASP to determine which rooms must be pathed through to reach a. given goal-state

It is hypothesised that by combining answer set programming methods and MDPs an effective path planning technique will be created. It is also hypothesised that by combining different ‘rewards’ for the MDP (hit-wall, and reached-goal) a suitable path will be planned for different movement models on a virtual robot.

Answer Set Prolog

Answer set prolog is a programming language that allows for knowledge representation and logical reasoning. Answer set prolog has a signature defined as the tuple Σ = *{O,F,P, V, S}.* These stand for objects, functions, predicates, variables and sorts. The version of answer set prolog used in this research was made compatible with the SPARC system (REFERENCE). SPARC requires ASP to be split into sorts, predicates, and rules. The Sorts section defines the variables (location, robot, and step), fluents (inertial or defined), and actions in 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)).

Which states that the robot has moved to location x. Inertial fluents can be altered by actions, and as such the inertial fluent ‘at’ can be altered by this action. Defined fluents cannot be altered by actions. The only defined fluent in this domain is adjacent, defining how rooms are connected.

Predicates define relations. Predicates used are: holds(fluent,step), occurs(action,step), success(), goal(step), something\_happened(step). For the ‘holds’ predicate, this defines that a fluent ‘holds’ at a given time step (i.e. it is true at a given time step. The same principle holds for occurs, but applies 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, for example, to define that a robot can not 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 is not 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.

Which 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.

The code written for this experiment has certain rules written in, and others which are written automatically by the code which splits a given map into rooms. (Room adjacency, goal, and start are all formed automatically. This is detailed in the methods section.

Method

The program written has four main parts: running the experiment; the Markov Decision Process; reading an inputted map; and integration with Answer Set Prolog. The program takes as input a map, a start coordinate, an end coordinate, 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 through only certain rooms

The map reading function reads an input black and white map and splits it into individual rooms which are each assigned a number. ‘Doors’ (holes in the wall of the room) are then 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. Two maps are required, one which contains only rooms and doors, and another which is the same but also contains additional obstacles. The map utilised can be seen in figure 1, as well as the start and goal point used for assessing the MDP.

A Markov Decision Process is used to create plans for each room of the map. Plans are created equal to the number of doors, with each plan using the centre of one of the room’s doors as a goal state. To calculate a plan, the MDP takes as input an array corresponding to the room (where 1 is a wall, and 0 is empty space), reward values for wall collision (RW), reaching the goal state (RG), and a value corresponding to probability of movement (Pr). The virtual agent used here when moving position, had a probability equal to Pr of moving in the intended direction, and a probability of (1-movement\_probability)/2 of going to either of the adjacent tiles instead. For each non-wall point on the map a value is assigned equal to largest expected reward of surrounding tiles. This is calculated by finding the expected reward in each possible direction (up, down, left, or right) by multiplying the reward from the intended spot by Pr, and the expected reward of the two adjacent tiles by Pw. 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 is ran iteratively, using the previous map as the map input for the next iteration. This is known as ‘Value Iteration’. To ensure a successful plan is formed, discounting is used. Discounting is explained here:

Where O is the outputted value, 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 the previous V, and if not changes it to O.

A plan is created for each door in each room when the program is first run, and then a final extra plan is created for the final goal state in whichever room this is located in.

For route planning, the virtual agent’s initial position is set in the first room as whatever the start coordinate is, and for all other required rooms 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 continues until the tile moved is the goal position, 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 be sufficient, and a route will not be found. To determine if this is the case the route planning function checks whether the next coordinate has previously been in the route, and if so the route is classed to have failed, skipping the current room and switching to path the next required room.

To decided which rooms need to be routed through, Answer Set Programming is utilised. A template ASP file is used and rules are added too 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 the first rule mentioned 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).

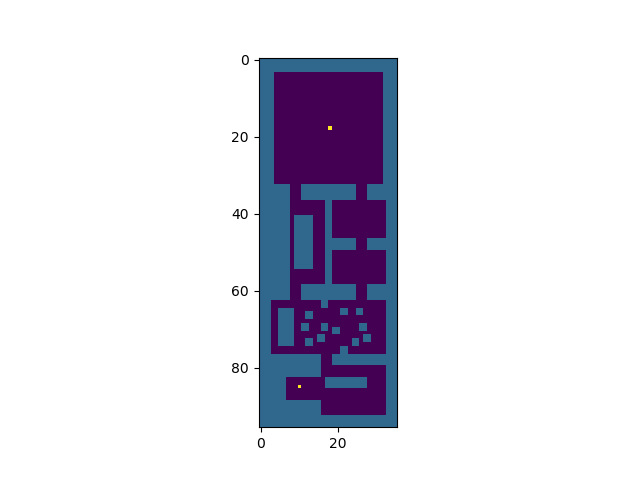
The first rule means robot is at x1 at time 0. The second rule means the goal is achieved at time I, if robot is at x2 at time I. The ASP file is then run to try and solve the optimal route. To optimise solving times, a maximum number of timesteps allowed is used. This starts at 1 and increases by 1 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 coordinate, a goal coordinate, 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 (EXPLAIN WHAT THISIS), 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 was 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. As such, a path in both directions from 0 to 5, and 5 to 0 will be assessed, and a minimal path would be that which does not go through rooms 2 and 3.

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, as is categorised by the program.



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 the map became similar values. An example of an MDP created from a discount of 1 can be seen below. Due 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.

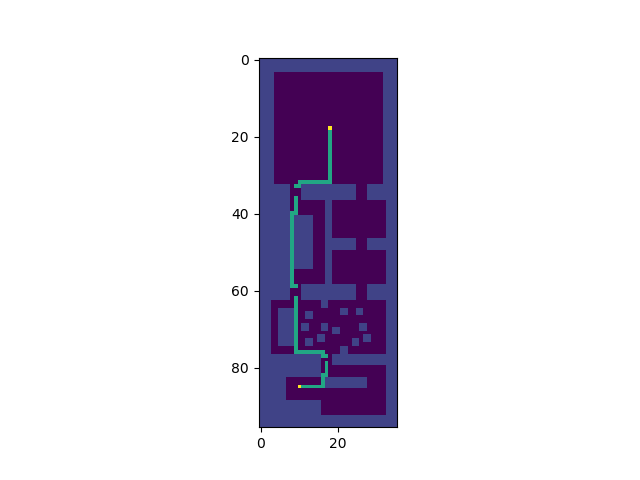


Figure 2. The optimal route planned for a Pr of 1. The yellow areas are the start and end points, and green points are the route planned.

For a Pr of 1 the optimal route was considered to be the most optimal route. All possible plans (excluding for a discount of 1) gave the exact same route, no matter what the parameters were set for. 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.

For a Pr of 0.8 routes which did not avoid small gaps (i.e. one tile wide gaps) were excluded. Routes which did not avoid these gaps were excluded, as well as routes which did take a minimal path but were closer to walls than other minimal routes. Minimal paths found 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.

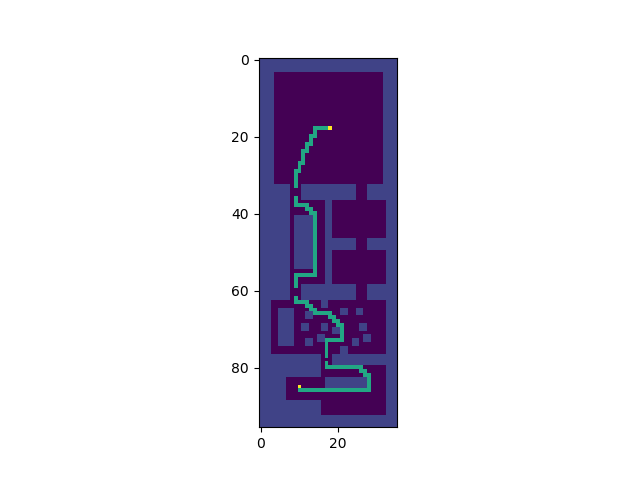


Figure 3. The optimal route planned for a Pr of 0.8. The yellow areas are the start and end points, and green points are the route planned.

For a Pr 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 criteria 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 when hit wall was -10, and reached goal was 10, and 100, and when hit-wall was -100 and reached goal was 10 or 1000. The routes planned despite the low movement probabilities still stuck close to the wall in most instances. There were 4 minimal plans, and the routes were 131 points long.

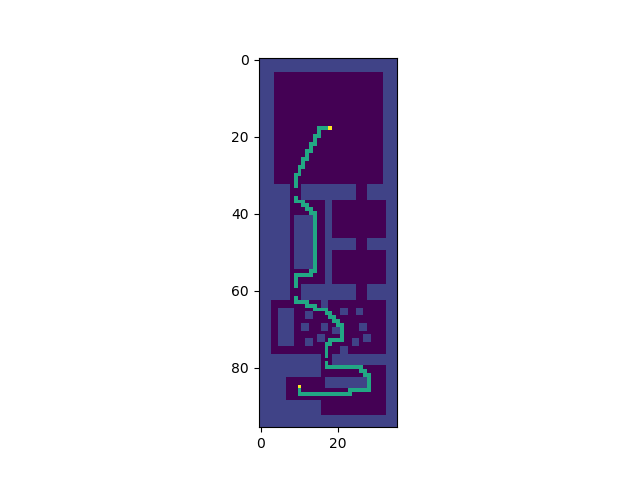


Figure 4. The optimal route planned for a Pr of 0.3. The yellow areas are the start and end points, and green points are the route planned.

Discussion

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.

For perfect movement (Pr of 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 the ASP code is suitable for planning paths through multiple rooms. For a Pr 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 particularly 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 maximised 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.

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 other 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.