**METAREASONING USING ANALOGY AND ITS APPLICATIONS TO ROBOTICS**

Adrián Lendínez Ibáñez 1400881

Supervisor: Doctor Renxi Qiu

Contents

[Problem definition: 2](#_Toc130991713)

[Research questions: 2](#_Toc130991714)

[Why experiences? Formal definition? 2](#_Toc130991715)

[A word about segmentation of experiences: 2](#_Toc130991716)

[Memory bank of experiences: 3](#_Toc130991717)

[Experience 1: Known situation 3](#_Toc130991718)

[Experience 2: Known situation 3](#_Toc130991719)

[Experience 3: Unknown situation 3](#_Toc130991720)

[Experience 4: Unknown situation 4](#_Toc130991721)

[Iterative estimation of experiences believes: 4](#_Toc130991722)

[Definition of MDP for choosing the right experience: 5](#_Toc130991723)

[Analogy calculation: 6](#_Toc130991724)

[State compression problem 7](#_Toc130991725)

[Find similarity of states: 7](#_Toc130991726)

[New experience as a linear combination of past experiences? 7](#_Toc130991727)

# Problem definition:

How to reuse previous experiences knowledge in a new/semi new situation using analogy?

**Assuming**: some historical data available at the time = 0. E1 & E2.

Robot does not know if it is in E1 or E2.

Verify if making good analogy, how fast convergence to the right experience (validation of theory).

 (If analogies are wrong, it should take more time to converge.)

# Research questions:

\* If newSittuation is either E1 or E2, this meta reasoning using analogy is always better than random just my pure logic. Prove by using simulation and many experiments averaging them.

\* The real interesting problem is if newSittuation is not E1 or E2 but resemblances somewhat to E1 or E2.

\* If newSittuation is a linear combination of E1 and E2, can the algorithm realize this and make better informed decision and create new E3?

\* **Big question:** what is the best way to get best analogy at the beginning?

# Why experiences? Formal definition?

An experience is a collection of states with their associated actions and the knowledge of which one should be executed. This knowledge is represented in this work as q-values and semantic rules. Q-values are specific whereas semantic rules and general.

An experience is the combination of a Q-Table with some semantic rules that procure generality. A Q-Table is a collection of states with their action weights (q-values). A semantic rule applies logical operators over some parameters of the world that may not be in the states of the Q-Table.

Without semantic rules, the number of experiences definitions with concrete, detailed and numerous states would be unmanageable. (Limitation of q-learning).

If so many detailed oriented and non-general experiences are generated, the probability of the robot to be in any experience is close to 0.

*An experience is not a full representation of the real-world situation. This is because assuming the robot will observe all parameters of the world and explore it all is unreasonable.*

Some experiences, because of their nature, may overlap one another. This means that 2 different experiences may produce the same successful result for the robot.

## A word about segmentation of experiences:

Real life is continuous and in this work experiences are discrete entities. The process by which experience segmentation is done is completely human guided for simplicity sake.

# Memory bank of experiences:

The collection of experiences the robot already has.

## Experience 1: Known situation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R | 10 | -100 | -100 | 5 | D100 |
| 10 | 15 | -100 | -100 | 5 | 80 |
| 10 | 15 | 15 | 15 | 15 | 60 |
| 0 | 18 | 20 | 30 | 50 | 55 |
| -10 | -10 | 10 | 10 | 34 | 45 |

All cells have a 100% lighting condition.

Experience1 90%  (Action success 95%)     Localisation (100%)

Action success: calculated by running the experiment many times and averaging.

## Experience 2: Known situation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| R | 10 | -100 | -100 | 0 | D100 |
| 10 | 15 | -100 | -100 | 0 | 80 |
| 10 | 10 | 0 | 0 | 0 | 60 |
| 0 | 18 | 20 | 30 | 50 | 55 |
| -10 | -10 | 10 | 10 | 34 | 45 |

All cells have a 0% lighting condition.

Experience2 10% (Action success 65%)   Localisation (100%)

Action success: calculated by running the experiment many times and averaging.

## Experience 3: Unknown situation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X (state x) | y |  |  |  |  |
| z |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Half the map (left hand side) has 0% lighting condition.

Experience3 10% (Action success 65%)   Localisation (100%)

Action success: calculated by running the experiment many times and averaging.

## Experience 4: Unknown situation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

(left hand side) has 0% lighting condition. Middle of map, 50% lighting condition and right 100%.

Experience4 10% (Action success 65%)   Localisation (100%)

Action success: calculated by running the experiment many times and averaging.

# Iterative estimation of experiences believes:

Initial data:

E1= Believe in Exp1 initial analogy P[E1(0)]

Under E1, the success rate of the Action A is = 95%

E2= Believe in Exp2 initial analogy P[E2(0))

Under E2, the success rate of the Action A is = 65%

A=Observed action success rate, to be collected during the iterations

Smoothing of the observed success rate using NLP: <https://en.wikipedia.org/wiki/Additive_smoothing>

Text

Description automatically generated

1st iteration

Given observed action success rate as

The believe of the Exp1 is updated as

New believes

The believe of the Exp2 is updated as

New believes

2nd iteration

Given observed action success rate as

The believe of the Exp1 is updated as

New believes

The believe of the Exp2 is updated as

New believes

Ith iteration

Given observed action success rate as

The believe of the Exp1 is updated as

New believes

The believe of the Exp2 is updated as

New believes

# Definition of MDP for choosing the right experience:

**Need to be defined: MDP\_reward for choosing right experience.**

**Strategies:**

-Change believe of E1 to E2 and act according to their policies dynamically or select a policy based on initial analogy and stick with it until the end.

State[0] = [map, robotPose, robot\_localization\_state]

E1 = Q\_table\_1

E2 = Q\_table\_2

Def Analogy(State[0], Experience):

Return analogical\_distance 🡪 P[E1(0)]

Changing experience reward -1000000000

Iteration 0:

Given State[0], E1 and E2.

P[E1(0)] = Analogy(State[0], E1) [Probab of E1 in t=0]

Historical + current state + Current\_confidence

P[E2(0)] = Analogy(State[0], E2) [Probab of E2 in t=0]

= MDP E1 [Prob of success of action in context E1] We mean prob of doing exactly the action even with noise. To calculate:

\* E1 = state1,state2,state3…stateN

\*P(MoveUp1|state) = (P(MoveUp1| state1) + P(MoveUp1| state2) + P(MoveUp1| stateN)) % N

\*P(MoveDown1|state) = (P(MoveDown1| state1) + P(MoveDown1| state2) + P(MoveDown1| stateN)) % N

\* P(Action| Ex1) = P(MoveUp1|state) + P(MoveDown1|state) + P(MoveLeft1|state) + …

= General given value. [Prob of success of action in any context] To calculate:

If the number of cells and parameters such us lighint are fixed, it cab be calculated accurately. Do we assume this?

(P(Action| Ex1) + P(Action| Ex2) + P(Action| ExN)) % N

Iteration 1:

Given State[1], E1 and E2.

=

Implement randomness in the terrain: smooth, raft.

Smooth map: expected 90% of terrain to be smooth.

Raft map: expected 90% raft.

# Analogy calculation:

Analogy is used to guess the most similar previous experience given some observations and initial state.

## State compression problem

Efficient way of representing state. Absolute position (fast). (Memory of humans)

## Find similarity of experiences:

How to compare similarity. If similar reward it may be the same situation. Estimate reward and have some estimation. If reward is low, maybe not do so much calculation. If the other way around.

### New experience as a linear combination of past experiences?

This is an assumption that needs to be made for the time been. If the robot cannot recreate the new experience based on past experience it means there are some parameters/variables that it does not know. If there is a map with slope and the robot never ever experience slope, it needs to create this concept and learn about it and its implications. Therefore, lets assume for now the new experience is a linear combination of past experiences.

## Reusing knowledge in new experiences that are linear combinations of previous past experiences.

Given a finite set of elements regarded as robot experiences

A new novel experience is a linear combination of when:

are known as the coefficients of the linear combination. For now, lets assume that the infinite set of new experiences is always a linear combination from elements of

An experience is formally defined as a mapping between a collection of states and the action to be taken. This actions are optimal to arrive to the destination. I.E Shortest path.

The is a matrix with elements that represent the given scenario. In this case, robot position, goal position and a map. Programmatically, lets define .  
  
state **=** [  
[EMPTY, OBSTACLE, GOAL],

[ROBOT, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],  
]  
ACTIONS **=** [UP, DOWN, LEFT, RIGHT]

knowledge(state) -> action

The knowledge function maps states with actions to be taken by a robot. For reinforcement learning, this can be a Q Table (). Let’s stay agnostic for how the state-action mapping is done for now. Inside the robot, there is some controller to execute the movement commands.

Transition(state, action)-> state’

If the transition function is executed until robot reaches destination:

GOAL\_REACHED = FALSE

previous\_state = state

WHILE (GOAL\_REACHED == FALSE)

Transition(previous\_state, action)-> state’

previous\_state = state’

check\_goal\_reach():

GOAL\_REACHED = TRUE

Eventually the robot reaches destination and the solution path is provided:

Solution **=** [[state, RIGHT], [state’, RIGHT], [state’’, UP]]

Path = [RIGHT, RIGHT, UP]

Visualizing for simplicity purpose:

e1 **=** [  
[EMPTY, OBSTACLE, GOAL],

[ROBOT, EMPTY, EMPTY],

[EMPTY, EMPTY, EMPTY],  
]

Here the experience is very self-contained. Let’s define using the visual notation.

e2 **=** [  
[EMPTY, EMPTY, GOAL],

[ROBOT, EMPTY, EMPTY],

[EMPTY, OBSTACLE, EMPTY],  
]

Here, but the paths are the same. Let’s assume an experiment in which the robot knows and . It now encounters a new experience which is a linear combination of and Let’s define

e3 **=** [  
[EMPTY, OBSTACLE, GOAL],

[ROBOT, EMPTY, EMPTY],

[EMPTY, OBSTACLE, EMPTY],  
]

A word about the addition process of experiences:

|  |
| --- |
| For the addition process of the experiences, some rules coming from the semantic interpretation of the experience are required. For example, EMPTY + EMPTY = EMPTY and not 2 EMPTY. The semantic understand is: given the same cell in a map with status empty in one experience and status empty in the other throughs an empty cell is the experience 3. EMPTY + OBSTACLE = OBSTACLE. For the robots additions, the robot is the same in both experiences so ROBOT + ROBOT = ROBOT and not 2 ROBOT. The same goes for the goal, the goal is fundamentally the same in both experiences, so GOAL + GOAL = GOAL and not 2 GOAL. |

Taking the additions rules into account, the following is true and corresponds to the definition of a linear combination.

The robot wants to reuse its previous experience in this new situation. How can it do it? First, the robot needs to understand that . Once it knows about this, there is one axion that can be applied for this context:

***1. Given an experience that is a linear combination of another two, and if those e1 and e2 have the same solution path (knowledge) then either of the two can be directly utilise over e3.***

e3 **=** [  
[EMPTY, OBSTACLE, GOAL],

[ROBOT, EMPTY, EMPTY],

[EMPTY, OBSTACLE, EMPTY],  
]

For verification purpose of axion 1, lets do another example. Given e3 as a linear combination of e1 and e2.

Diagram

Description automatically generated

There is one thing to note before moving on. Applying this axion assuming that the robot knows fully about e3. This either means it has already explorer e3, it has partially explorer e3 and made assumption about linear combination or all information about e3 is giving to it in some way. For application of the second axion, notice that it is not necessary to know all about e3 before start making a decision. This is especially convenient since realistically a robot at time t will only perceive a certain amount of the map.

This was the easy scenario. Now let’s have two experiences with different topology and solution paths. How can we reutilise their knowledge in a linear combination of both?

The second axion says that **2.** ***Given an experience (e3) that is a linear combination of another ones (e1 & e2),* there is always a local analogy in either e1 or e2 at time t to e3 to which local knowledge of e1 or e2 can be directly applied. These local similarities allow for geometrical operations such as multiplication, addition, reflection etc. If such is required, the knowledge also needs to be geometrically transformed with the same operation.**

In the upcoming example depiction of knowledge transformation is provided for clarification purpose. Given e3 as a linear combination of e1 and e2 like in the figure below:

A picture containing timeline

Description automatically generated

Applying the second axion over the robot initial position, e1 local = e3 local in the yellow region. Since the robot cannot move diagonally, using knowledge of e1 at time t=0 is possible.

Diagram

Description automatically generated

According to e1, robot will move 1 to the left. The process is repeated again.

Diagram

Description automatically generated

In this case, locally e1 is again equal to local e3. Applying state-action knowledge of e1, robot will move 1 to the left.

Diagram

Description automatically generated

After this, e2 locally is equal to e3. Robot moves two down according to state-action of e2. Note two steps actions were done at a time but can be done in two different iterations. Now the following state is interesting as by default, without any geometrical operation, there is no local similarity over e1 or e2.

Diagram

Description automatically generated

As a consequence, the robot needs to find a local similarity over e1 or e2 with a geometrical transformation. In this case, local e2 (highlighted in yellow below)is a geometrical reflection of e3.

Diagram

Description automatically generated

Even if in e2 there is empty, it is actually a robot occupied cell as marked by the path, so don’t confuse. In this case, applying reflection over the action-state of e2 is move 1 right. Finally, robot reaches destination. The robot has successfully used its previous knowledge to solve a new situation.

Diagram

Description automatically generated with medium confidence

Programmatically, the robot is switching from one believe to another constantly via local analogies. It can also happen that the robot knows e3 and while exploring, it will never switch to other experience as e3 is the correct one. In the following chapters it will be explored in more detail the iterative estimation of experience believes and the process of switching from experiences along with its consequences. This is modelled after an MDP over experiences switching. Notice that there is also another MPD, the task MDP that will select the robot movement action according to the knowledge of the experiences. In this case, there are 2 MDP’s one for meta reasoning and another for low level robot movement commands.

What is good about the second axion approach is that it is iterative. It updates its believes with sensor feedback, like traditional estimation filters like Kalman or particle filters.

There is, however, one problem with the axion 2 as it is. The robot can observe fully e3 at time t=0. This is not realistic. Additionally, e1 and e2 and fabricated. Conveniently selected to accommodate the examples. Nevertheless, the real scenario will have the robot know many experiences, hundreds or thousands. But, most importantly, if the robot cannot fully observe e3 at time t = 0, how can it guarantee the linear combination assumption? The linear combination assumption is responsible for the additions of local experiences of e1 or e2 to always arrive to the goal. If the robot selects some experience that is locally similar but not linear combination, the robot will not reach to the destination. Here is an example, the robot starts assuming it is in e1 because of local similarities to new experience e3 (yellow marking). At this point the robot believes in:

e1 **=** [  
[EMPTY, EMPTY, OBSTACLE],



[ROBOT, EMPTY, EMPTY],



[EMPTY, GOAL, EMPTY],  
]



e3\_real **=** [  
[EMPTY, OBSTACLE, GOAL],



[ROBOT, EMPTY, EMPTY],



[EMPTY, EMPTY, OBSTACLE],  
]



When the robots moves to the right and then down, it will notice the goal was not reached. e1 is not a linear combination of e3. There is no point in trying to find some e2 that added to e1 will reach e3, this will never occur. e1 needs to be fully discarded. To the robot, there maybe hundreds of experiences that are locally similar to e3 at time t = 0, but there is no guarantee they will arrive to the destination. There is no guarantee of a linear combination situation. It is still better than random walk, but not enough. **So here comes a critical aspect of the solution, proper experience candidates selection at time t = 0, and also at time t 1,2,3…**

Lets introduce some more complex semantics to ground the experiment. The goal is to find a child. Goal == child. And there is now a new object of type toy. If the robot looks into the historical experiences (lets assume the robot now has 100 experiences available) it may recognise some patterns and create some *high level extra knowledge*. Robot has detected that in all experiences where there is a toy, the child is always next to it (either to the left, right, up or down). This new knowledge was produced by reasoning mechanism known as **induction**. Inductive knowledge is not always reliable but it is easy to generate. It goes from the particularities to the generality. To the robot this is a heuristic to guide the search for experience selection. If at time t = 0 robot detects a toy, it will only select as experience candidates the ones that satisfy the heuristic, moreover, it will restrict the goal search space. Lets do an example:

and make analogy (projection with the hopes of continuity) that if there is a toy, the

E3\_expected **=** [  
[EMPTY, OBSTACLE, OBSTACLE],

[ROBOT, EMPTY, EMPTY],

[OBSTACLE, GOAL, EMPTY],  
]

E2 **=** [  
[EMPTY, OBSTACLE, EMPTY],

[ROBOT, EMPTY, EMPTY],

[OBSTACLE, GOAL, EMPTY],  
]

==========================================

Scenarios:

PMDP: states are infinite. You can assume saying there are limited states.

**How much tolerance the robot has? Depending on the task ahead.**

How to do the Verification?

Unknown Situation 1   (Action success ??%):

Skill == expiernces.

How to select skill:

Updating confidence based on fuzzy reasoning.

S(1,2,3)

S\_policy(1,2,x,y,5) 2 reward -1

Unknown Situation 2

Calculate Q-table using Bellman equation and epsilon greedy algorithm

1. Define formually in maths the 1+1

Decompose and recombine experience (human). Verify and experience repeat.

E = Q table

action1, action2, action3

State 1 : 1 2 2.3

-Table 3

-Circle detection & MDP.

1. First step: use historical data and safety constraints to decide

E1: Prob(70)% [Confidence: 30%]

E2: Pro(50) % [Confidence: 30%]

E3: 25% [Confidence: 30%]

For each believe in exp we should have a confidence level.

MDP Task completion: Reward for completing task:

MDP for building up world model: Reward for maximizing confidence:

To switch: safety + confidence + probability. => reward for switching -1000

Confidence of none of the above: new E4

Assumptions: world is static for now. Static distribution of randomness (Smoothness of terrain of each cell).

\* Action points:

1. Create Exp1 Smooth terrain. 🡪 generate Q table Ex1 Skill Smooth terrain.
2. Create Exp Raft terrain .
3. Calculation P[E1(0)] of Prob Exp with Analogy: Historical data + current state + Current\_confidence
4. Implement Iterative estimation of experiences believes (Bayes equations)
5. MDP for building up world model.
   1. Reward for choosing experiences for each action step.
   2. Action: Switch or not switch.
   3. States: Prob high/low + conf high/low

Table for update iteration of MDP.

1. For exp.
2. Confidence.

\* Evaluation:

-Given a policy of 90% smooth and another one with 10% smooth. Select the best policy for 75% of smoothness using the 2 MDPs. (By assumption 75% is closer to 100% meaning First policy should do better.) Need to evaluate properly.