Learning to solve Sliding Puzzles using Reinforcement Learning

Umr Barends

18199313

Report submitted in partial fulfillment of the requirements of the module
Project (E) 448 for the degree Baccalaureus in Engineering in the Department of
Electrical and Electronic Engineering at Stellenbosch University.

Supervisor: JC Schoeman

October 17, 2020

Acknowledgements

I would like to thank my dog, Muffin. I also would like to thank the inventor of the incubator; without him/her, I would not be here. Finally, I would like to thank Dr Herman Kamper for this amazing report template.

Plagiaatverklaring / *Plagiarism Declaration*

1. Plagiaat is die oorneem en gebruik van die idees, materiaal en ander intellektuele eiendom van ander persone asof dit jou eie werk is.

   Plagiarism is the use of ideas, material and other intellectual property of another's work and to present is as my own.

2. Ek erken dat die pleeg van plagiaat 'n strafbare oortreding is aangesien dit 'n vorm van diefstal is.

   I agree that plagiarism is a punishable offence because it constitutes theft.

3. Ek verstaan ook dat direkte vertalings plagiaat is.

   I also understand that direct translations are plagiarism.

4. Dienooreenkomstig is alle aanhalings en bydraes vanuit enige bron (ingesluit die internet) volledig verwys (erken). Ek erken dat die woordelikse aanhaal van teks sonder aanhalingstekens (selfs al word die bron volledig erken) plagiaat is.

   Accordingly all quotations and contributions from any source whatsoever (including the internet) have been cited fully. I understand that the reproduction of text without quotation marks (even when the source is cited) is plagiarism

5. Ek verklaar dat die werk in hierdie skryfstuk vervat, behalwe waar anders aange-dui, my eie oorspronklike werk is en dat ek dit nie vantevore in die geheel of gedeeltelik ingehandig het vir bepunting in hierdie module/werkstuk of 'n ander module/werkstuk nie.

   I declare that the work contained in this assignment, except where otherwise stated, is my original work and that I have not previously (in its entirety or in part) submitted it for grading in this module/assignment or another module/assignment.

Studentenommer / *18199313*	Handtekening:
Initials and surname / *U.Barends*	Datum / *October 17, 2020*

Abstract

English

The English abstract.

Contents

List of Figures

List of Tables

Nomenclature

Variables and functions

$p(x)$ Probability density function with respect to variable x.

Acronyms and abbreviations

RL   Reinforcement Learning

SARSA   State-Action-Reward-State=Action

Q-learning   Q value learning

MDP   Markov Decision Process

Chapter 1

Introduction

1.1. Background

In the modern world robots are becoming more and more part of our daily lives and society. For example the Xiaomi house cleaning robot. Generally in robotics a manipulator (eg. an arm) is used to manipulate an object in the environment.

It is easier to bfirst simulate the robots behavior in a more simple environment which is why we will solve a similar and smaller problem as a step to a complete solution. By doing sp we can save expenses, as it is costly to test a robot in reality while simulation is much cheaper.

The sliding puzzle is a game with a history going back to the 1870's. The aim of the game is to place a shuffled set of 15 numbered tiles into ascending order by sliding and swapping the black/blank tile piece into one of its neighboring tiles. An example of a puzzle being solved is shown in 1.1.

1.2. Problem Statement/ Research Aim

In this project we solve sliding puzzles using reinforcement learning, where the same algorithm can then later be applied to the robotics problem of moving in an environment. Seen in Figure 1.1 is an example 4x4 puzzle.
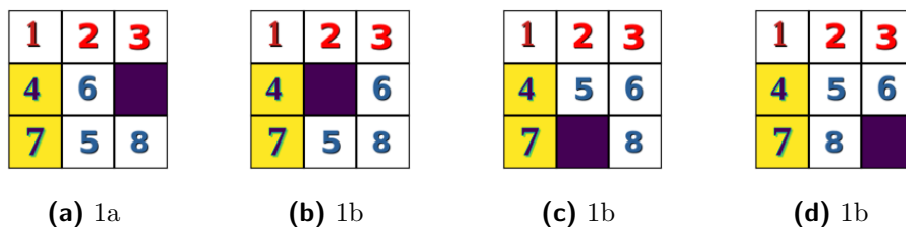

(a) 1a   **(b)** 1b   **(c)** 1b   **(d)** 1b

Figure 1.1: Figures of a 3x3 sliding puzzle being solved

1.3. Project Objectives

- Solve a 2x2 sliding puzzle using Reinforcement learning

- Solve a 3x3 puzzle using Reinforcement learning

- Puzzles must solve in a reasonable amount of time

1.4. Scope

Although there are many methods of solving a problem in RL, in this project we only look at two methods. These are SARSA (State-Action-Reward-State-Action) and Q-learning.

Typically for problems with large state spaces neural networks are used in conjunction with the Reinforcement Learning techniques, to save computational time. However for this project we used a certain method to overcome the state space limitation which will be discussed later.

1.5. Document Outline

To follow along the report, we first layout the required knowledge in 2 for understanding the topics covered in the following chapters. Then next we look at a paper with a similar problem to this reports as well as their solution.

Then in 3 we describe in detail the mathematical background theory which forms the basis for Reinforcement Learning. In Chapter 4 we then detail the methods of Reinforcement Learning that will be used to solve our puzzle problem.

Chapter 5 lays out the overarching view of the software system, broken down into functional blocks. We then in Chapter 6 we discuss experiments and results and verify that our solution works. Finally with Chapter 8 the report is concluded.

Chapter 2

Literature Review/Study/Related work

2.1. Introduction

Reinforcement learning is a method of learning what to do by linking states to actions with a numerical reward incurred for every state-action pair. The final goal of RL is to find a path of states and actions with the maximum amount of reward. [3]

2.2. Solving sliding puzzle using value iteration

In this section a few terms will be used which are fully described in the theory chapters 3 and 4. These terms include value iteration and policy iteration.

A 4x4 puzzle solution is investigated in [4]. The method they use is a variation of value iteration. The amount of states in a 4x4 puzzle is approximately $10\hat{1}3$ elements. It was found in [4] that value iteration is indeed a feasible approach if the value iteration algorithm is altered so that the only a subset of the entire state space is used sequentially until a solution is found.

This reducing of the state space is similar to what will later be done in this report. However the method we use to reduce the state space is by having our algorithm be guided somewhat by how a human would solve a puzzle, by doing the edges first. Additionally that we use a different reinforcement learning method known as policy iteration.

Solutions to sliding puzzles can also been found by using search algorithms such as A* and IDA* [5].

Chapter 3

Markov Decision Processes

3.1. Markov Decision Processes (MDP's)

To model reinforcement learning problems we use dynamic systems theory, specifically using incompletely-known MDP's (Markov decision processes). Most of RL problems can be described using MDP's. In Mathematics a MDP is a stochastic, discrete time control process.

What that means is that the process is essentially partially controlled and partially random. MDP's depend mainly on a few variables which are, states, actions, state transition probability ,reward and a discount factor [3]. These variables can be denoted in a 5-tuple as:

$$(S, A, P_{ss'}^a, R_s^a, \gamma)$$

where

- S is a finite set of states

- A is a finite set of actions

- $P_{ss'}^a$ is a matrix of probabilities with $P_{ss'}^a = P(S_{t+1} = s' | S_t = s, A_t = a)$

- R_s^a is the immediate reward after transitioning from state s to s' using action a. Where $R_s^a = E[R_{t+1} | S_t = s, A_t = a]$

- $\gamma \in [0,1]$ is the discount factor applied to the reward

For a state to be Markov it needs to satisfy the following condition:

$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, ..., S_t]$$

This means that the current state is required to contain all the information of the previous states. For a MDP all states must be Markov.

We now define the definition which is the total discounted reward from time-step t,

named the return G_t where:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... \tag{3.1}$$

$$= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3.2}$$

The discount $\gamma \in [0,1]$ determines the present value of future rewards. For $\gamma = 0$ the return G_t only depends on the current reward that can be obtained in the next step R_{t+1}. Which can be said to be "short-sighted". While for $\gamma = 1$ the return depends on all the rewards that are projected to be obtained until the process terminates. This can be said to be "far-sighted". The larger γ is the more the rewards of later steps closer to the terminating state affects the return. [3] What we can also note is that using a discount factor makes the return finite because R_{t+k+1} is finite and:

$$\sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}$$

which is finite for $\gamma \in [0,1]$.

We now define another important required definition, namely the state value function v(s) where:

$$v(s) = E[G_t | S_t = s]$$

Which in word mean that the value state function v(s) is the expected return given that the agent is in state s at time-step t. We now decompose v(s) so that it becomes a recursive function as follows:

$$v(s) = E[G_t | S_t = s] \tag{3.3}$$

$$= E[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s] \tag{3.4}$$

$$= E[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) | S_t = s] \tag{3.5}$$

$$= E[R_{t+1} + \gamma G_{t+1} | S_t = s] \tag{3.6}$$

$$= E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s] \tag{3.7}$$

This means that v(s) only depends on R_{t+1} the reward it can incur in the next step as well as $v(S_{t+1})$ the discounted state function at time t+1. What we have just defined is known as the Bellman Equation. [3]

The probability an agent takes a certain of action given it is in a given state is defined as a policy $\pi(a|s)$. It is defined as:

$$\pi(a|s) = P[A_t = a, S_t = s]$$

Then we can now define the state value function following policy π as:

$$v_\pi(s) = E_\pi[G_t|S_t = s]$$

Additionally we define the action-value function also known as a q-function as:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]$$

The q-function is defined as the expected return taking action a from state s thereafter following policy π. What can be noted is that v(s) is a prediction of what the value for being in a certain state is. While q(s,a) is the value for being in a certain state and taking a action, which has to do with control compared to v(s) which has to do with planning.

Once again we can decompose the state value and also action value functions to be in the form of the Bellman Equation in equation 3.7. This results in the Bellman *Expectation* Equations:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{3.8}$$
$$= \sum_{a' \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')) \tag{3.9}$$

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{3.10}$$
$$= R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s')q_\pi(s', a') \tag{3.11}$$

The Bellman Expectation Equations can also then be put into matrix form from equation 3.9 for fast vector calculation as:

$$v_\pi = R^\pi + \gamma P^\pi v_\pi \tag{3.12}$$

Which has the solution:

$$v_\pi = (1 - \gamma P^\pi)^{-1} R^\pi \tag{3.13}$$

Equation 3.13 is a fast and concise formula which is easily implemented in code, but has the drawback of only working for small state spaces. This is due to the fact that the calculation required the entire transition matrix P^π to be loaded into memory at every calculation step.

We now define what is known as the optimal policy, which is essentially a set of state

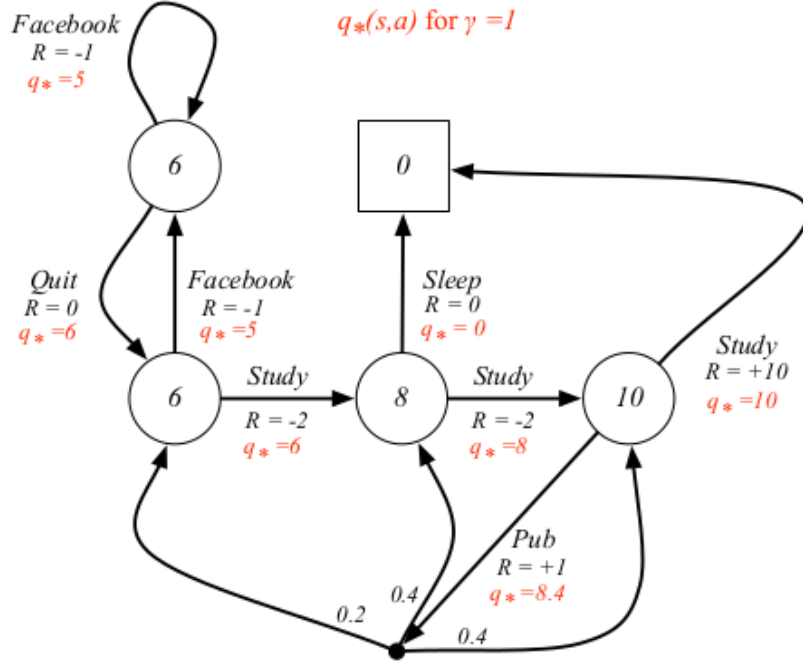# Example: Optimal Action-Value Function for Student MDP

Figure 3.1: Graph of an example MDP showing how q values work [1]

and action pairs which lead to the maximum $q_\pi(s, a)$ as:

$$\pi_*(a|s) = \begin{cases} 1, & \text{if a} = \underset{a \in A}{\arg\max}\, q_*(s, a) \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

There always exists an optimal deterministic policy for any MDP [3]. We only need to know $q_*(s, a)$ to know the optimal policy. This is important because the optimal policy describes to us how the agent must move to maximize its rewards, which is precisely the goal of RL. Figure 3.1 shows an example MDP displaying how $q_*(s, a)$ works. The red path lines indicate the optimal path/policy in Figure 3.1.

We now finally define what is know as the Bellman *Optimality* Equations which is a specific form of equation 3.7 and 3.9 which are:

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s') \tag{3.15}$$

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a') \tag{3.16}$$

What equation 3.15 represents is the maximum reward that can be obtained in state s. While equation 3.16 represents the maximum reward that can be obtained by taking action a from state s. The difference between the two is that equation 3.15 is just a passive

prediction of what value (according to reward that can be obtained) a state represents. While equation 3.16 is a calculation of the value of being in a state based on after a action 'a' is selected.

Chapter 4

Dynamic Programming and Reinforcement Learning

4.1. Introduction

4.2. Dynamic Programming

4.2.1. Introduction

Dynamic programming is a method to solve problems by using a divide and conquer approach. That is to say a problem is broken into smaller sub-problems and then solved. There are two properties that must be met for a problem to be solvable via Dynamic Programming: Optimal substructure and Overlapping sub-problems. [1]
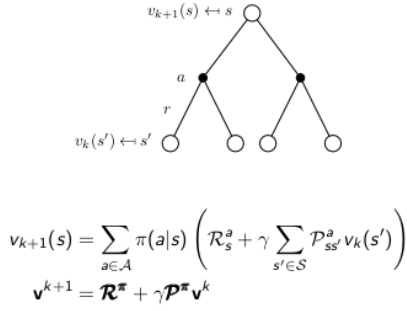
- Optimal substructure requires that the solution be able to be decomposed into sub-problems.

- Overlapping sub-problems requires that the sub-problems occur many times over, meaning that the solution can be cached and reused.

Markov decision processes satisfies both of these conditions. The Bellman equation provides a recursive decomposition which fulfills the Optimal substructure property. While the value function caches and reuses solutions satisfying the Overlapping sub-problems condition. [1]

4.2.2. Iterative Policy Evaluation

In order to evaluate the value that a certain policy has, we use a method called *Iterative Policy Evaluation*. Figure 4.1a shows the mathematics which describes Iterative Policy Evaluation in both summation and vector form. What the diagram at the top of Figure 4.1a represents is the value function being calculated for an action 'a' to an intermediary state represented by the black dot. The environments reaction is then also accounted for

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_k(s') \right)$$

$$\mathbf{v}^{k+1} = \mathcal{R}^{\pi} + \gamma \mathcal{P}^{\pi} \mathbf{v}^k$$

(a) Iterative policy Evaluation [1]
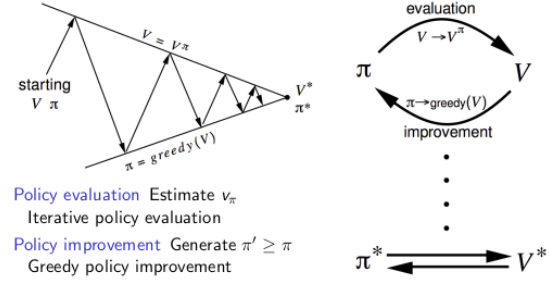
(b) Greedy Policy iteration visualisation [1]

Figure 4.1: Policy Iteration [1]

from the intermediary state to state s' resulting in reward r. The environments information is contained in the transition probability matrix P^{π}.

We then use state s' as the new start state s and repeat the same calculation. The value function is updated continuously until the difference between v^k and v^{k+1} is determined negligible.

4.3. Reinforcement Learning

4.3.1. Introduction

There are two separate uses for Dynamic Programming in Reinforcement Learning. These are prediction and control which both have (S, A, P, R, γ) as input.

- The prediction method uses (S, A, P, R, γ) and a policy π as input, while it outputs a value function v_{π}.

- The control method only takes (S, A, P, R, γ) as input and outputs the optimal value function v_* and optimal policy π_*.

Maybe insert a flow diagram illustrating difference and similarity between prediction and control

These two methods of Dynamic programming are implemented via what is known as value iteration and policy iteration respectively.

4.3.2. Policy Iteration

Figure 4.1b shows how Greedy Policy Iteration works. A greedy policy is one which when followed, results in the maximum value function v_{π}. It is known that policy iteration converges which can also be seen in the diagram in Figure 4.1b on the top left. [3]

As is stated in Figure 4.1b policy iteration works by continuously iterating between two steps, namely policy evaluation and policy improvement.

For a given policy π we improve the policy using the following two steps:

Step one is *Evaluating* the policy π using equation 3.9:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

Step two is *Improving* the policy by acting greedy with respect to v_π to obtain a new policy π':

$$\pi' = greedy(v_\pi)$$

What this means is that we obtain a value function by using iterative policy evaluation where after we update the policy. In this way we will always converge to the optimal policy π_* and value function v_* described by equation 3.14 and 3.15 respectively.

4.3.3. Value Iteration
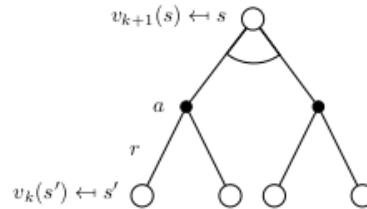
Figure 4.2: Value Iteration [1]

Figure 4.2 shows the mathematics that describes *Value Iteration*. It is very similar to iterative policy evaluation in Figure 4.1a. Iterative Policy Evaluation calculates the value function for a policy π which is an entire set of actions given states.

While Value Iteration calculates the maximum value function for a single action only. In this sense Value Iteration is a special case of Iterative Policy Evaluation.

Can later place in appendix proof of policy iteration always converging. slide 19 (16/42) of David Silvers slides on Dynamic Programming

Chapter 5

System Design

insert software diagram here

5.1. Thing 1

To solve the puzzle I could have used monte carlo learning, ... I chose SARSA and Q-learning because ...

5.2. Thing 2

5.3. Methods

- Measure reward-episodes graphs

Chapter 6

Applying reinforcement learning to sliding puzzles

6.1. SARSA figs

6.2. Q-learning figs

6.3. Discussion

Chapter 7

Solvability of a NxN sliding puzzle

info in this chapter to still be integrated somewhere else into report if needed. Else will leave it out

7.1. General puzzle description

Let us assume that we have an NxN puzzle, then we have NxN number of blocks. We can represent the puzzle as an NxN array, then we stack the array into a one dimensional array of 1 x (N*N). For example see the 4x4 puzzle in Figure 7.1 we have a 1 x 16 array as: Array = (12,7,8,13,4,9,2,11,3,6,15,14,5,1,10). Before we describe the conditions for a sliding puzzle to be solvable, we first define the term "inversion". Assuming the the first index of the 1xN 2 array starts at the left top corner (valued 12) in Figure 7.1, and that it runs from [0,(N*N)-1]. Then an inversion occurs when Array[index] ¿ Array[index+1] where index is an arbitrary integer between 0 and N*N-1. Hence in Figure 7.1 we have a total: sum of inversions(Array) = 11 + 6 + 6 + 8 + 3 + 5 + 1 + 5 + 1 + 2 + 4 + 3 + 1 + 0 = 56.

Figure 7.1: Example of a sliding puzzle

7.2. Conditions for solvability

Even and odd sized boards are analysed separately (where size = N).

For odd sized boards where N is odd we have the puzzle only being solvable if and only if the boards has an even number of inversions. The proof for this can be deduced by looking at Figure 2 and noting that for every switch of the blank block we have an even change in the sum of inversions of the board. [2]
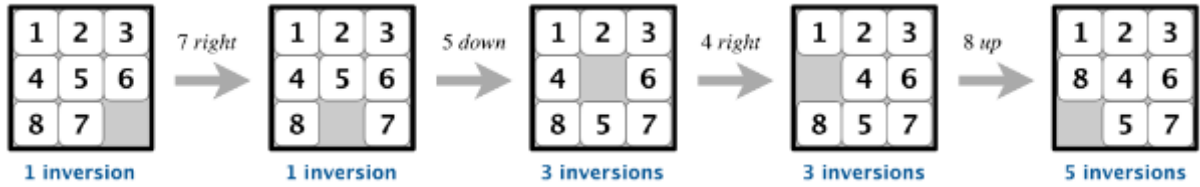

Figure 7.2: Odd boards with change in blank piece only having even inversion change [2]

For even sized boards where N is even we have the board solvable if and only if the number of inversions plus the row of the blank square is odd. This is illustrated in Figure 3.
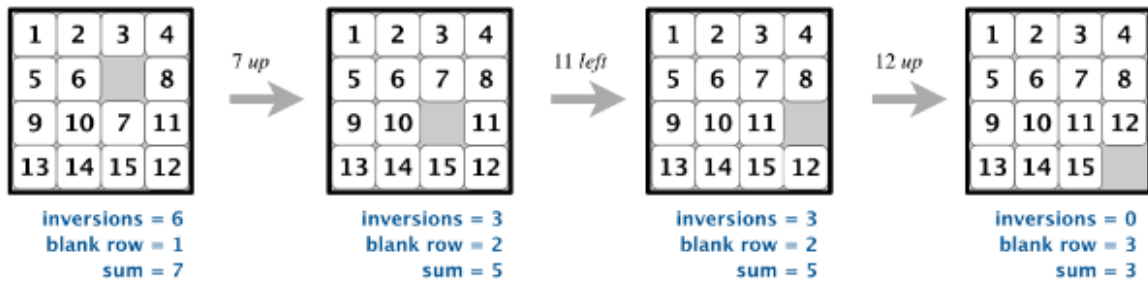

Figure 7.3: Even board solvability [2]

Half of all puzzle configurations are unsolvable. [6] This means that we only have N! / 2 configurations that are solvable for an NxN board. This was proven using parity in the paper in [6]. Sliding puzzles can be solved relatively quickly with today's processing of computers for puzzles for example an 5x5 puzzle was solved in 205 tile moves in 2016. [7]

The issue more so lies in finding the shortest path to solving a puzzle. This specific problem of solving with the least amount of tile moves of a sliding puzzle has been defined as NP (non-deterministic polynomial-time) hard. NP hardness is are problems that are as least as hard as NP. Where in computational complexity theory NP (non-deterministic polynomial-time) is a has a solution with a proof variable to be in polynomial time by a deterministic Turing Machine. A Turing machine is a mathematical model defining an abstract machine which manipulates symbols according to a set of rules. [8]

In simpler terms a problem is NP if it can be solved within a time that is a polynomial function of the input. For instance if we define the time to solve a problem as 'T' and the input data as 'D'. Then as long as T = polynomial function (D) then a problem is NP.

Chapter 8

Summary and Conclusion

8.1. Summary

Objective 1 ...

    Objective 2 ...

    Objective 3 ...

    So what? Relates back to problem and background sections in introductions.

8.2. Future work and reccomendations

Bibliography

[1] D. Silver, "Ucl course on rl," 2015. [Online]. Available: https://www.davidsilver.uk/teaching/

[2] P. U. F. of Computer Science, "8puzzle assignment." [Online]. Available: https://www.cs.princeton.edu/courses/archive/spring18/cos226/assignments/8puzzle/index.html

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning An Introduction second edition.* The MIT Press, 2018.

[4] H. M. Bastian Bischoff, Duy Nguyen-Tuong and A. Knoll, "Solving the 15-puzzle game using local value-iteration," 2013.

[5] R. E. Korf, "Depth-first iterative-deepening: an optimal admissible tree search. artificial intelligence 27," pp. 97–109, (1985).

[6] W. E. Johnson, Wm. Woolsey; Story, "Notes on the "15" puzzle," *American Journal of Mathematics*, vol. 29, no. 2(4), p. 397–404, 1879.

[7] D. of the Cube Forum, "5x5 sliding puzzle can be solved in 205 moves," 2016. [Online]. Available: http://cubezzz.dyndns.org/drupal/?q=node/view/559

[8] Minsky, *Computation: finite and infinite machines.* Prentice Hall, 1967.

Appendix A

Project Planning Schedule

This is an appendix.

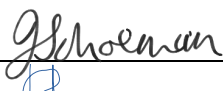# Appendix B

Outcomes Compliance

This is another appendix.

Appendix C

Student and Supervisor agreement

Agreement between skripsie student and study leader regarding mutual responsibilities

Project (E) 448, Department of Electrical and Electronic Engineering, Stellenbosch University

Student name and SU#:
Umr Barends
18199313

Study leader:
Mr JC Schoeman

Project title:
Learning to solve Sliding Puzzles using Reinforcement Learning

Project aims:
Generally in robotics a manipulator (eg. an arm) is used to manipulate an object in the environment. It is normally easier to first simulate the robots behavior in a more simple environment. In this project we try to solve a sliding puzzle using reinforcement learning, where the same algorithm can then later be applied to the robotics problem.

1. It is the responsibility of the student to clarify aspects such as the definition and scope of the project, the place of study, research methodology, reporting opportunities and -methods (e.g. progress reports, internal presentations and conferences) with the study leader.
2. It is the responsibility of the study leader to give regular guidance and feedback with regard to the literature, methodology and progress.
3. The rules regarding handing in and evaluation of the project is outlined in the Study Guide/website and will be strictly adhered to.
4. The project leader conveyed the departmental view on plagiarism to the student, and the student acknowledges the seriousness of such an offence.

Signature — study leader:

Signature — student:

Date:
5 August 2020

(Upload this form to SUNLearn)

Figure C.1: Student and Supervisor agreement