Technical University of Crete

School of Electrical & Computer Engineering

Alternating Policy Iteration

An Analysis and Future Directions

Diploma Thesis

Author:
Athanasios N. Bacharis

Thesis Committee:
Associate Prof. Georgios Chalkiadakis

Associate Prof. Michail Lagoudakis

Assistant Prof. Athanasios Aris Panagopoulos

A thesis submitted in partial fulfillment of the requirements for the degree of Diploma in Electrical and Computer Engineering

October 2019

Περίληψη

Οι Μαρκοβιανές Διαδικασίες Απόφασης (ΜΔΑ) αποτελούν ένα ισχυρό μαθηματικό μοντέλο για λήψη αποφάσεων υπό αβεβαιότητα. Έχουν εφαρμοστεί σε διάφορα επιστημονικά πεδία, όπως τα οικονομικά, η επιχειρησιακή έρευνα, η λήψη ιατρικών αποφάσεων, και η ρομποτική. Στη βάση της, η λύση μιας ΜΔΑ για την απόκτηση μίας βέλτιστης πολιτικής είναι υπολογιστικά ακριβή, και το πρόβλημα επιδεινώνεται στις μεγάλες διαστάσεις (δηλαδή σε μεγάλους χώρους κατάστασης - ενέργειας). Με βάση τα παραπάνω, έχει προταθεί στη βιβλιογραφία πληθώρα προσεγγιστικών μεθόδων για την αντιμετώπισης τη χωρικής και χρονικής πολυπλοκότητας υπολογισμού λύσεων ΜΔΑ.

Μία ενδιαφέρουσα προσέγγιση προτάθηκε το 2015, από τους Παναγόπουλο-Χαλκιαδάκη-*Jennings*. Η εν λόγω προσέγγιση χρησιμοποιεί μία εναλλασσόμενη μέθοδο βελτιστοποίησης αποφάσεων σε υποχώρους κατάστασης - ενέργειας. Παρόλο που η ιδέα εργασίας σε υποχώρους λύσεων ενός προβλήματος βελτιστοποίησης δεν ήταν καινούργια, αυτός ο αλγόριθμος ήταν ίσως ο πρώτος που πρότεινε μία τέτοια προσέγγιση στα πλαίσια της ΜΔΑ. Η ίδια δημοσίευση επίσης αναδεικνύει την επιτυχία μίας τέτοιας προσέγγισης στο χειρισμό ενός φωτοβολταϊκού συστήματος παρακολούθησης ηλίου. Ωστόσο, η ίδια δημοσίευση δεν ξεκαθαρίζει πώς αυτός ο καινούργιος αλγόριθμος κλιμακώνεται σε σχέση με το μέγεθος του προβλήματος, ούτε πως συγκρίνεται με τις κλασσικές προσεγγίσεις επανάληψης πολιτικής και επανάληψης ανταμοιβής· και δε μπορεί να χρησιμοποιηθεί σε περιβάλλοντα τα οποία δεν αφήνουν την εκτέλεση των υπολογισμένων ενεργειών έπειτα από την βελτιστοποίηση στις διαχωρισμένες διαστάσεις. Το τελευταίο συνδέεται εννοιολογικά με καταστάσεις που έχουμε φαινόμενα "παραποίησης" (*aliasing*) πληροφορίας. Η παραποίηση πληροφορίας εμφανίζεται σε διάφορα επιστημονικά πεδία, όπως οι τηλεπικοινωνίες και η ρομποτική, και αντιστοιχεί στην απώλεια πληροφορίας έπειτα από τη μείωση των διαστάσεων ενός προβλήματος προκειμένου να προσεγγιστεί η λύση του.

Ως εκ τούτου, σε αυτή τη διπλωματική εργασία παρουσιάζουμε μία νέα παραλλαγή του αλγορίθμου της εναλλασσόμενης επανάληψης πολιτικής που επιλύει τα προαναφερθέντα θέματα *aliasing*, και παρέχουμε συγκρίσεις με τους αλγορίθμους επανάληψης πολιτικής και επανάληψης ανταμοιβής. Δείχνουμε εμπειρικά ότι ο προτεινόμενος *Aliasing Aware Alternating Policy Iteration* (*AAAPI*) αλγόριθμός μας μπορεί να συγκλίνει στις βέλτιστες λύσεις (πολιτικές), με την παρουσία φαινομένων *aliasing* πληροφορίας. Επίσης, η υπολογιστική πολυπλοκότητα αυτού του αλγορίθμου είναι σε άμεση συσχέτιση με την ένταση της *aliasing* πληροφορίας. Σε περιβάλλοντα όπου τα φαινόμενα *aliasing* δεν είναι τόσο έντονα, ο *AAAPI* συγκλίνει γρηγορότερα σε σχέση με τις μεθόδους επανάληψης πολιτικής (*policyiteration*) και επανάληψης τιμών (*valueiteration*)· αλλά σε περιβάλλοντα με υψηλό βαθμό *aliasing* πληροφορίας, όπως ο Λαβύρινθος, ο ρυθμός σύγκλισης του *AAAPI* πέφτει

δραματικά. Μία επιπλέον συνεισφορά της εργασίας μας είναι η διατύπωση μία πιθανής επέκτασης του $AAAPI$ σε πολυπρακτορικά περιβάλλοντα.

Abstract

Markov Decision Processes (MDPs) constitute a powerful mathematical model for decision making under uncertainty. They have been used widely in a number of application areas such as economics, operation research, health care and robotics. In their fundamental form, solving an MDP to derive its optimal policy is computationally expensive, and the problem is only exacerbated in its high dimensions (i.e., in large state-action spaces). To this end, a number of approximate solution methods have been proposed over time, tackling time and space complexity in various ways.

An interesting approach has been proposed in 2015, by Panagopoulos et al, that utilizes an iterative optimization method to optimize over state-action sub-spaces. Although the idea of iteratively optimizing over sub-spaces, is not new in optimization theory, this algorithm was perhaps the first to propose such an approach in the context of MDPs. The same paper also illustrates the success of such an approach in controlling a solar tracking system. Nevertheless, that work does not illustrate clearly how this new algorithm scales along with problem size, nor how it compares with typical policy iteration or value iteration approaches; and could not be used in environments that do not allow the execution of the actions computed after optimization in each separate dimensions. Intuitively, this corresponds to situations where we have information aliasing phenomena. Information aliasing is a concept which appears in many scientific fields, such as telecommunications and robotics, and describes the loss of information due to dimensionality reduction.

As such, in this thesis we provide a novel variant of the alternating policy iteration algorithm that resolves the aforementioned aliasing issues, and provide a comparison with policy iteration and value iteration. We show empirically that Aliasing Aware Alternating Policy Iteration (AAAPI) can converge to the optimal solutions (policies), in the presence of information aliasing phenomena. Also, the computational complexity of this algorithm is directly related to the intensity of information aliasing. In environments where information aliasing is not intense, AAAPI converges faster than policy iteration and value iteration; but in high-aliasing environments like the maze-grid, the AAAPI convergence rate is substantially reduced. Finally, we provide a discussion on a possible AAAPI multi-agent extension.

Acknowledgments

After these years of studying at the Technical University of Crete, I would like to express my gratitude to the people who were by my side all these years and helped me in my achievements.

First of all, I would like to thank my advisors, *Prof. Georgios Chalkiadakis* and *Prof. Athanasios Aris Panagopoulos*, for their guidance and continuous advising. I am also grateful to the third member of the committee *Prof. Michail Lagoudakis*, for his kindness and for letting me join *Kouretes Laboratory*, where I had my first exposition to the scientific field of reinforcement learning.

Last but not least, I am deeply thankful to the members of my family and my friends, from Chania and from Athens, who supported me in, not only the good times, but the bad ones also for many years in my life.

Contents

List of Figures

Chapter 1

Introduction

In the field of Artificial Intelligence (AI), one of the main problems that occur, is related to the need of autonomous agents to take sequential decisions, in order to complete a task or achieve a goal. Sequential decision making refers to the process, which agents have to decide an action (or reaction) to take in the current state or their environment, so as to proceed to the next one. Usually, these actions serve a certain purpose, e.g. a goal, and the agent needs to plan to achieve it. In figure 1.1 we can see a simple planning problem, where we have an agent, which needs to reach the goal via a certain plan.

Planning thus is the procedure followed to compute future actions, depending on the situation (states), in order to complete a task. Planning is studied in decision analysis, operations research, control theory and economics. In real life environments, planning is not always a deterministic procedure, it contains uncertainty (stochastic process) due to future parameters that can change the conditions of decision making. These parameters can be actions from other agents that cooperate or compete with our agent (economics) or general conditions that depend on the environment.

These planning problems, which contain stochastic process, can be modeled as Markov decision processes (MDPs)[3] and can be analyzed using the techniques of decision theory. Decision theoretic planning (DTP)[12] is an extension of planning; it allows to model problems in which actions have uncertain effects, the decision maker (agent) has partial knowledge of the world, and the solutions come from resource consumption of the Markov model.

Apart from decision theoretic planning, reinforcement learning (RL)[34] can be seen as another method for plan learning. Reinforcement learning uses the technique of interactive learning. In this technique, the agent interacts with the environment, by acting inside it, and getting a positive or negative feedback. Using this feedback, the agent can decide whether or not its actions are effective, forming its plan in the process. Reinforcement

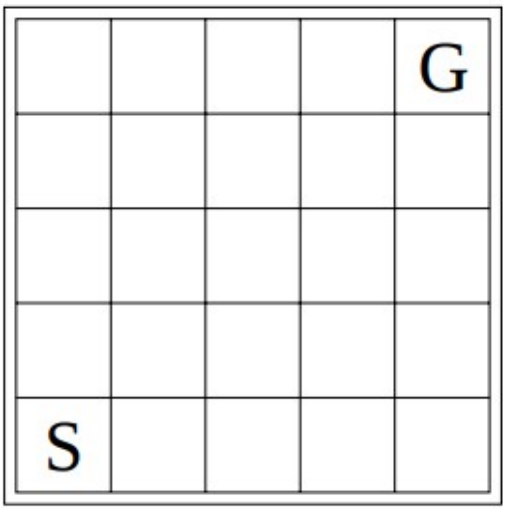

Figure 1.1: A 5x5 Grid world with an agent starting at S trying to reach goal G via an 'optimal' path

learning problems can be modeled as Markov decision processes with unknown components, and thus the algorithms which are used to solve these MDPs differ from those used in DTP.

1.1 Motivation

The mathematical model, Markov decision processes [3], it is widely known and it has been applied in many scientific field, especially those who require decision making. In the first steps of planning and reinforcement learning, dynamic programming algorithms (policy iteration and value iteration) came to solve MDPs and produce decision making results. Although the powerfulness of the model, the higher the dimensions grows, the higher the time and space complexity grows with it. To encounter this complexity which makes our model not feasible, approximations and decomposition techniques came to deal with the problem.

In 2015, the paper towards optimal solar tracking [17] proposes an alternating policy iteration schema that utilizes a long known alternative optimization technique in the context of policy iteration for optimal solar tracking. This technique decomposes the solar tracking space in x and y dimensions, in order to tackle the curse of dimensionality phenomena (when the dimensionality increases, the volume of the space increases so fast that computation becomes infeasible in time), and solve the sub-problems separately. In this case, decision theoretic planning can decompose the dimen-

sions and work separately, since the separate plannings of the space could be easily combined and produce a global solution. A method like this, is easy in use and can work fast and efficient, as the paper illustrates, for solar tracking.

This approximation technique it is not easily implemented in every environment. There are environments where the decomposition of their space into sub-problems, and the combination of their solutions is not feasible. These kind of environments usually have information aliasing phenomena, which appears in dimensionality reduction attempts and describes the loss of information. It is essential to improve this method, in order to be applied in more complicated environments with information aliasing phenomena. Also, a more compact analysis of the complexity and the efficiency of the algorithm is required for further applications.

1.2 Contributions

In this thesis we provide a novel alternating policy iteration (AltPI) algorithm and its convergence rate. This algorithm created in order to encounter the problem of information aliasing, which we face due to decomposition process. Also, for the need of performance comparison and optimality results, we implemented two classical decision making algorithms, Policy iteration and Value iteration.

The environment we used to test and evaluate our algorithms is a grid-world with obstacles ($walls$). This environment provides a good generalization for our algorithms, since many static environments, proper for decision making problems-algorithms, can be reduced in a grid-world. Also, with the obstacles in the grid-world and the decomposition method, which our algorithm uses, we create information aliasing phenomena. An interesting parameter is that the more the obstacles in the grid-world the more intense is the aliasing phenomena. The worst case with the most information aliasing phenomena is a grid-world with the maximum obstacles (maze).

As we have mentioned, information aliasing appears in dimensionality reduction (i.e. decomposition), creating difficult conditions for our algorithm to perform due to loss of information in the lower dimensions. Alternating policy iteration can work under these conditions and can, not only, produce optimal results (policies) in out environments, but also outperform, in time complexity, the two classic algorithms in some environmental cases. We noticed that AltPI works faster in grid-worlds where information aliasing phenomena is not so intense, rather than environments with many obstacles. This property of our algorithm, makes clear the cases/environments where it can be used.

Finally, this algorithm idea can be extended in a multi-agent environ-

ment. These types of environment contain more than one agents, that can cooperate or compete. The planning methods in these situations require reinforcement learning (like Q-Learning) and not DTP algorithms, since RL can work with more than one agent and less information about the environment, while DTP is proper for one agent and static environments where all the information is available.

Chapter 2

Theoretical Background

In this chapter we will discuss decision theoretic planning (DTP), reinforcement learning, and information aliasing. We will provide a detailed introduction to Markov decision processes (MDPs).

Firstly, we will build the problem of planning formalizing the concepts defined to solve MDPs. These concepts include properties of the Markov model, equations that are used for planning, and properties of these equations. We will also discuss the first approaches of planning, which include dynamic programming (DP) techniques, and introduce two classic DP algorithms.

Finally, we will provide an introduction to reinforcement learning (RL), which can be considered as an extension of DTP since it is used to compute a planning for the agent, but with less information regarding the environment. RL handles environment; where there are parameters that require estimation. We will introduce a classic RL algorithm, Q-learning. Following that, we will discuss the information aliasing problem.

2.1 Decision-Theoretic Planning

The classical planning problem is that of producing a sequence of actions that guarantees the achievement of certain goal when applied to a specified starting state [6]. The problem of decision-theoretic planning involves the design of plans or policies in situations where the initial conditions and the effects of actions are not known with certainty, and potentially actions must be traded against one another to determine an optimal course of action [12]. A DTP problem can be viewed as a problem of optimal stochastic control, for this reason Markov decision process have been proposed as a semantic and computational framework to formulate DTP. This model allows the formulation of actions with stochastic effects and the specification of states or objectives of differing importance.

2.1.1 MDP & Markov Property

Markov Decision Process (MDP) is a discrete time stochastic process that was first introduced by Bellman and it is useful for studying optimization problems solved via dynamic programming or other reinforcement learning techniques. A planning task that satisfied the Markov property (we will define what Markov property is below) is called a Markov Decision Process [34].

A particular MDP is defined by a five attribute tuple, {S, A, P, R, γ}, where S stands for the set of possible states and A stands for the set of actions. The transition model P is actually a probability distribution over the next states and it is dependent from the current state and the taken action in that state: $P(s^{'}|s,a)$. The reward model depends on three variables, the current state, the next state and the taken action: $R(s,a,s^{'})$. This value model can be stochastic and the value which returns is usually scaled with the previous probability distribution (transition model P) and the discount factor $\gamma \in (0,1)$. The discount factor γ is a number, which weights the expected value return. In practice, the discount factor is used to make uncertain the decisions in the next instants of the environment.

Given the above parameters, we can fully determine our Markov decision process. The discount factor, the probability and reward tables are dynamics that are determined from the environment of the problem, and can be changed given the conditions. In figure 2.1, we can see a classic example of a recycling robot. It has two states, high and low, three actions, search, wait and recharge, the rewards given the action, the current and the next state, and finally the transition probabilities. It is noticeable that from every state the transition probabilities, given the action, always sum to 1.

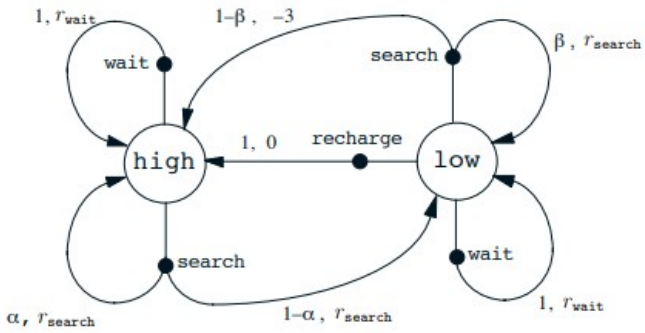

Figure 2.1: MDP: Transition graph example of a recycling robot [34]

Now that we have formulated Markov decision processes, we can talk about an important property that characterizes them, the Markov property.

Assuming we have a finite number of states and reward values a general environment will respond at a time $t+1$ to the action taken at time t. In the general environmental case, the response may depend on everything that has happened earlier. So the dynamics can be defined by specifying the complete probability distribution:

$$Pr\{R_{t+1} = r, S_{t+1} = s^{'} | S_0, A_0, R_1..., S_{t-1}, A_{t-1}, R_t, S_t, A_t\} \tag{2.1}$$

for all $r, s^{'}$, and all possible values of the past events: $S_0, A_0, R_1..., S_{t-1}$, A_{t-1}, R_t, S_t, A_t. If the state signal has the Markov property, then the environment's response at $t+1$ depends only on the state and action representations at t, in which case the environment's dynamics can be defined as:

$$p(s^{'}, r|s, a) = Pr\{R_{t+1} = r, S_{t+1} = s^{'} | S_t, A_t\} \tag{2.2}$$

for all $r, s^{'}, S_t$ *and* A_t. In other works, a state signal has the Markov Property, and is a Markov state, if and only if 2.1 is equal to 2.2 for all $r, s^{'}$. In this case, the environment and task as a whole are also said to have the Markov property.

2.1.2 Bellman equation

As we have discussed, almost all planning or reinforcement learning problems have as an ultimate goal the estimation of a value function-function of states. This function estimates how good the states of the problem are for our agent.

Via the value function, the agent can determine a policy, so in each state it can decide which action to choose in order to get better value. First lets define the term policy. The policy is actually a mapping from each state, $s \in S$, to an action, $a \in A(s)$, with probability $\pi(a|s)$ of taking that action a when in state s. So we would like to choose the best policy for each state. To accomplish this, we can combine the value function and the policy definitions and see that the more accurate the value function we can estimate is, the better the policy.

To calculate value functions, we have to estimate the value of a state s under a policy π, $v_\pi(s)$. This is the expected return when starting in s and following π thereafter, so formally we can define $v_\pi(s)$ as:

$$v_\pi(s) = \mathbb{E}_\pi[G_t | S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s] \tag{2.3}$$

where G_t is the discounted reward value, $\mathbb{E}_\pi$ is the expected value of a random variable given that the agent follows the policy π, R is the reward

model, γ the discounted factor, while t is any time step. The function $v_\pi(s)$, is the state-value function for policy π.

Similarly, we can define the value of taking action a in state s under a policy π, $q_\pi(s,a)$, as the expected return starting from s, taking action a and then following policy π:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \quad (2.4)$$

q_π is called the action-value function for policy π.

One fundamental property of the value function used throughout reinforcement learning and dynamic programming is that it satisfies particular recursive relationships. For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)[r + \gamma\, \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2}|S_{t+1} = s']] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \end{aligned} \quad (2.5)$$

The equation 2.5 is the Bellman equation for v_π [4]. It expresses a relationship between the value of a state and the values of its successor states. The Bellman equation 2.5 is a weighting average for all the possibilities. In other words, the value of a state s is equal to the discounted value of the next state. This value function v_π refers to the unique solution of the Bellman equation.

For a finite MDP, we can define an optimal policy in the following way. Value functions define a partial ordering over policies. A policy π' is better or equal to a policy π, if its expected return is greater than or equal to that of π for all states. In other words, $\pi' \geq \pi$, if and only if, $v_{\pi'}(s) \geq v_\pi(s)\ \forall s \in S$. The optimal state-value function, we denote it as v_* and defined as:

$$v_*(s) = max_\pi u_\pi(s), \forall s \in S \tag{2.6}$$

Optimal policies also share the same optimal action-value function, denoted q_*, and defined as:

$$q_*(s,a) = max_\pi q_\pi(s,a), \forall s \in S \; and \; a \in A(s) \tag{2.7}$$

Defining these, v_* is the value function of a specific policy π_*. So, through the property of the value functions, it must satisfy the self-consistency condition from the Bellman equation 2.5 we saw above for state values. This is the Bellman equation for v_*, or the Bellman optimality equation 2.6 and it can be written in a form without any reference to a specific policy [4]:

$$\begin{aligned} v_*(s) &= max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= max_a \, \mathbb{E}_{\pi_*}[G_t | S_t = s, a = A_t] \\ &= max_a \, \mathbb{E}_{\pi_*}[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, a = A_t] \\ &= max_a \, \mathbb{E}_{\pi_*}[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} | S_t = s, a = A_t] \\ &= max_a \, \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_*(S_{t+1}) | S_t = s, a = A_t] \\ &= max_{a \in A(s)} \sum_{s',r} p(s', r | s, a)[r + \gamma v_*(s')] \end{aligned} \tag{2.8}$$

The equation 2.8 is the Bellman optimality equation for v_*. For q_*, the Bellman optimality equation is:

$$\begin{aligned} q_*(s,a) &= \mathbb{E}_{\pi_*}[R_{t+1} + \gamma max_{a'} q_*(S_{t+1}, a') | S_t = s, a = A_t] \\ &= \sum_{s',r} p(s', r | s, a)[r + \gamma max_{a'} q_*(s', a')] \end{aligned} \tag{2.9}$$

2.1.3 Policy Evaluation & Policy Improvement

Dynamic programming (DP) is a collection of algorithms that can solve and Markov decision processes optimally, building progressively on previously estimated solutions. Having determined our planning purpose, MDPs, value function (given the Bellman equations), we can now consider how we can evaluate our given policy π and how to improve our policy π every time given the value function $v_\pi(s)$.

Firstly, we examine again the Bellman equation for state-value function 2.5. If we look more carefully into this equation, we will notice that the

value function is estimated under a policy π. Also it calculates a weighed sum of the values for the next state, under the examined policy. In other words this equation calculates the expected reward for every state, of our MDP, under the policy π. In the book of Sutton and Barto [34], they provide an iterative way (algorithm 1) to estimate the value of a policy, in solved by Eq 2.5.

Algorithm 1 Iterative Policy Evaluation [34]

Input π, *the policy to be evaluated*
while $\Delta < \theta$ **do** ▷ *Where* θ *a small positive number*
  $\Delta \leftarrow 0$
  for each $s \in S$ **do**
    $v \leftarrow V(s)$
    $v(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s^{'}, r)|s, a)[r + \gamma V(s^{'}]$
    $\Delta \leftarrow \max \Delta, |v - V(s)|)$
Output $V \approx v_\pi$

Furthermore, having a way to evaluate our policy π for our MDP and eventually estimate a state-value function for our agent, a step for finding better actions is required. To this end a greedy method, in the spirit of dynamic programming, has come to propose an optimal solution. Suppose that we have to compute a state-value function $v_\pi(s)$ for a deterministic policy π. We would like to know, for some states s, whether or not the policy should be changed in order to choose an action $a \neq \pi(s)$ deterministically. The value of this behaving is:

$$\begin{aligned} Q(s,a) &= \mathbb{E}_{\pi_*}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s, a = A_t] \\ &= \sum_{s',r} p(s^{'}, r|s, a)[r + \gamma v_\pi(s^{'})] \end{aligned} \tag{2.10}$$

The criterion to this equation is whether this is greater or less than $v_\pi(s)$. If it is greater, it means that selecting action a, when in state s and then follow the policy π, is a better option.

The above criterion, indicates the policy improvement theorem. If π and $\pi^{'}$ are any pair of deterministic policies such that, for all $s \in S$,

$$q_\pi(s, \pi^{'}) \geq v_\pi(s) \tag{2.11}$$

Then the policy $\pi^{'}$ must be as good as π, or even better. Thus, with the new policy $\pi^{'}$we should obtain greater or equal expected return from all states $s \in S$:

$$v_{\pi'}(s) \geq v_\pi(s) \tag{2.12}$$

If there is a strict inequality of 2.11 at any state, then there must be strict inequality of 2.12 in at least one state. Thus, if $q_\pi(s,a) > v_\pi(s)$, then the changes policy is indeed better than π. The proof of the policy improvement is straightforward. It's a greedy way to update your policy π, based on the update of the estimated state-value function $v_\pi(s)$. The proof is on the book [34] pages 94-96.

Having understood the two above steps, policy evaluation and policy improvement, two classic algorithms created by using them, within the family of Dynamic Programming (DP)[4]. DP may not be practical for very large problems, but compared with other methods for solving MDPs[14][5], DP methods are actually quite efficient. The worst case time complexity for a DP method, is polynomial in the number of states and actions. If we denote n the states space, and m the actions space, we can find a polynomial function of n and m, that the complexity of our DP algorithms will be analogous to it.

2.1.4 Policy iteration

Policy iteration [14], is one of the first dynamic programming algorithms. The idea behind the algorithm is that once we improve our policy π, by using v_π to find a better policy π', then we can compute our new $v_{\pi'}$, with the new policy π', and improve it again to find an even better π''. This method can be characterized as a greedy one, since it chooses (greedily) every time the best actions. The greedy algorithms of DP are widely known and are used to solve (optimal) problems. We can thus obtain a sequence of monotonically improving policies and value functions as we see in the figure 2.2.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_{*},$$

Figure 2.2: Policy Iteration steps till convergence [34]

Each policy is guaranteed to be strict improvement over the previous one, unless it has converged to optimal value function u_π^* and thus policy π^*. This happens because in a finite MDP (in this thesis we deal with finite a grid worlds) has a finite number of policies, where the number of possible combinations is m^n (m actions space, n states space). So this process must converge to an optimal policy and optimal value function in a finite number of iterations [34].

Below, we have illustrated the pseudo-code of the algorithm:

Algorithm 2 Policy Iteration [34]

initialize policy π ▷ Initialization
initialize $V(s) = 0 \qquad \forall s \in S$
while π *is not stable* **do**

for each $s \in S$ **do** ▷ Policy Evaluation step
$V(s) \leftarrow \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$

for each $s \in S$ **do** ▷ Policy Improvement step
$\pi(s) \leftarrow \text{argmax}_{a \in A} \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$

return V, π

As we see, in this algorithm the steps policy evaluation and policy improvement are iteratively executed until the process converges.

2.1.5 Value Iteration

Noticing policy iteration, we see that each of its iterations involves policy evaluation, which may itself be a protracted iterative computation requiring multiple sweeps through the state set. If policy evaluation is done iteratively, then convergence exactly to v_π occurs only in the limit.

The policy evaluation step of policy iteration can be truncated in several ways without losing the convergence guarantees of it. One important case is when policy evaluation is stopped after one sweep. The above procedure is known as value iteration. It can be written as a particularly simple backup operation that combines the policy improvement and truncated policy evaluation steps.

$$\begin{aligned} v_{k+1}(s) &= max_a \, \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = a] \\ &= max_a \sum_{s', r} \pi(s', r | s, a)[r + \gamma v_k(s')] \end{aligned} \tag{2.13}$$

for all $s \in S$. For arbitrary v_0, the sequence v_k (where k is the number of v estimations) can be shown to converge to v_* under the same conditions that guarantee the existence of v_* [34].

We have to consider how value iteration method terminates. Value iteration, like policy evaluation, formally requires an infinite number of iterations to converge exactly to v_* [34]. In practice, this method stops when the value function change by only a small amount. Algorithm 3 illustrates the value iteration method with the above termination condition.

Algorithm 3 Value Iteration [34]

initialize policy π ▷ Initialization
initialize $V(s) = 0 \qquad \forall s \in S$
choose $\theta < 0.01$.

while $\delta < \theta$ **do** ▷ Policy Evaluation
  for each $s \in S$ **do**
    $V \leftarrow V(s)$
    $V(s) \leftarrow \max_\alpha \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$
    $\delta \leftarrow \max(\delta, |V - V(s)|)$
return $\pi(s) = \text{argmax}_\alpha \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$ ▷ Policy Improvement

In Value Iteration, the evaluation part is changed, as we described above. It is as efficient as the normal policy evaluation method and although it seems to be better than policy iteration method, since in the first look seems to execute less steps, the complexity of this algorithm is similar to that of policy iteration [34].

2.2 The Reinforcement Learning Problem

The idea that we learn by interacting with our environment is probably the first to occur to us when we think about the nature of learning. Learning from interaction is a foundational idea underlying nearly all theories of learning and intelligence [34].

Reinforcement learning is the process by which the agent interacts with its environment, evaluates its feedback and continues with "better" (updated) actions. This method can be considered as a planning process for agents in uncertain environments. RL problems can also be modeled as MDPs having all the properties we mentioned in the previous section, but now the transition model is unknown, meaning that the agent can learn directly from raw experience without a model of the environment's dynamics; and also the reward model is potentially unknown.

2.2.1 The Agent's Environment Interface

The problem of reinforcement learning is meant to be a straightforward framing of the problem of interactive learning to achieve a goal [34]. The thing it interacts with, comprising everything outside the agent, is called the environment. By interacting continually, the agent selects actions and the environment responds to those actions by presenting new situations to the agent. To make the agent understand whether he is making good

actions, the environment returns except from the new states, a reward for them also. To make things clearer in Figure 2.3, we provide an example of how the agent interacts in a realistic problem.

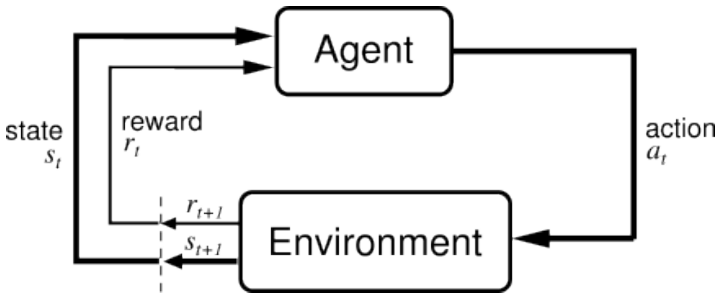

Figure 2.3: Interaction between agent and his environment [34]

More specifically, the agent interacts with its environment in a discrete time steps, $t = 0, 1, 2, 3, ..., .$ At each time step t, the agent selects an action $a_t \in A$ at its current state $s_t \in S$, then the environment, one time step later, responds to the agent with a reward $r_{t+1} \in R$ and a new state s_{t+1}, due to the action that the agent has made.

At each time step, the agent chooses an action from its policy π, which is a mapping from its states to the actions. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run.

2.2.2 Reinforcement Learning Methods

In standard reinforcement learning model, an agent is connected to its environment via perception and action, as described above. The agents behavior, should choose actions that tend to increase the long run sum of values of the reinforcement signal. It can learn to do this over time through the interaction guided by a wide variety of methods (algorithms).

As we have mentioned Markov decision processes is a mathematical model that is been used to describe a variety number of problems. Like in DTP, MDPs are able to model reinforcement learning problems, but with not the exact same way, since DTP and RL are different planning methods. The MDP in DTP is usually a four dimensional tuple $\{S, A, R, P\}$, the states S, the actions A, the reward table R and the probability transition table P. In RL things are different, there are cases where the problem, by its nature, does not give us all the required informations. These cases are model free problems, where MDPs do not have the transition probability table or the reward function and hence, the algorithms are beyond the perspective of dynamic programming.

Monte Carlo (MC) methods are ways of solving the RL problem through sampling [34]. With MC methods we use episodic tasks, which are created from our policy. With these episodes we evaluate the actions-states to estimate our value function $Q(s, a)$ and update our policy in order to create new episodes. Now, as we have described in previous sections, the value of a state is the expected return—expected cumulative future discounted reward—starting from that state [34]. To estimate the value function from experience, we can simply average the observed returns after visiting a state. The more observations we get the more accurate the value function will be. This technique implies all Monte Carlo methods. Some of these methods are first-visit and every-visit Monte Carlo which relies on the number of times we estimate the value of the state we visit during the episodic task. The other two are on-policy and off-policy Monte Carlo methods, which refer to the ways (what policy do we use) we generate our episodes, every time at the end of an epoch. Finally, both first-visit and every-visit MC converge quadratically to the expected values $q_\pi(s, a)$ as the number of visits (or first visits) to each state–action pair goes to infinity [34].

Considering a central idea to reinforcement learning, we should think of the temporal-difference (TD) learning [34]. TD learning combines Monte Carlo and dynamic programming (DP) ideas. From the aspect of Monte Carlo methods, TD method learns from raw experience without the need of the environment's dynamics. While from the aspect of DP, TD methods updates its estimations without the final outcome. An advantage of TD methods is, as we referred, that they do not use any of the environment's dynamics [34].

All these methods DP, MC, TD are fundamental for reinforcement learning. They have been the basis for many years in RL problems and cover a variety of cases. Based on them many algorithms have been proposed like PI and VI (DP algorithms), which we have seen in the previous section and many more. We now present one of the most important ones, which will also concern us in this thesis in Chapter 6.

2.2.3 The Q-Learning Algorithm

One of the most important breakthroughs in reinforcement learning was the development of an off-policy TD control algorithm known as Q-learning (Watkins)[36]. Q-learning is a form of model-free reinforcement learning and it can also viewed as a method of asynchronous dynamic programming. It provides agents with capability of learning to act optimally in Markovian domains by experiencing the consequences of actions, without requiring them to build maps of the domains.

Its simplest form, one-step Q-learning, is defined by:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \operatorname{argmax}_{a'}(Q(s^{'}, a^{'}) - Q(s,a)) \tag{2.14}$$

where $\{s, a, r, a^{'}\}$ is the tuple that describes our MDP, while the α parameter refers to the learning rate. If the learning rate is large then at every time step we make more greedy decisions/steps whereas if it is low then we take small steps.

In this case, the action-value function Q, approximates q_*, the optimal action-value function, without considering the policy π. This dramatically simplifies the analysis of the algorithm and enabled early convergence proofs [34]. Although, the policy π is still used to determine which state–action pairs are visited and updated. However, for the right convergence we require that all pairs will continue to be updated. This requirement is needed in order to find optimal policies and any general case must require it [34]. Under this assumption and the usual stochastic approximation conditions on the sequence of step-size parameters, Q has been shown to converge with probability 1 to q_* [34].

The algorithm of Q-learning is described below:

Algorithm 4 Q-Learning [36]

initialize $Q(s,a), \forall s \in S, a \in A$, arbitrarily, and $Q(terminal-state, .) = 0$ ▷ Initialization
repeat(for each episode)
  initialize S
  repeat(for each step of episode):
    Choose a from S using policy derived from Q
    Take action a, observe $R, s^{'}$
    $Q(s,a) \leftarrow Q(s,a) + a[r + \gamma \operatorname{argmax}_{a'} Q(s^{'}, a^{'}) - Q(s,a)]$
    $s \leftarrow s^{'}$
  until s is terminal
until episodes end

This algorithm has been altered into many variants in order to run faster and optimal in complex and realistic environments. Some of these variants are Deep Q learning [30], Double Q learning [13], Deep Double Q [32] and more.

2.3 Information Aliasing

Information aliasing describes the loss of information. Information aliasing phenomena appear in many scientific fields usually in data processing and

especially when dimensionality reduction is involved. In telecommunications, and more specifically in signal processing, aliasing is an effect that causes different signals to become indistinguishable when sampled [8]. It also refers to the distortion or artifact that results when the signal reconstructed from samples is different from the original continuous signal.

Apart from signal processing, aliasing exists in other fields also. In computing, for example, it describes a situation in which a data location in memory can be accessed through different symbolic names in the program. Thus, modifying the data through one name implicitly modifies the value associated with all aliased names, which may not be expected.

Finally, in problems where dimensionality reduction and decomposition happens, aliasing also appears. In this case, lower dimensions may not be able to represent and contain all the information from the higher dimensions. Thus, while working in lower dimensions approximations and prediction for the loss of information is required, since we do not want to lose optimality.

Aliasing phenomena thus occur in many cases, and the need to overcome this problem is intense. We will discuss in Chapters 4 and 5, how aliasing appears in our work and how we were able to counteract this phenomena with success. In fact, the problem we faced can be described as a perceptual aliasing problem.

2.3.1 Perceptual Aliasing

In robotic mapping which is related to computer vision and cartography, the goal for the robot is to be able to construct or use a map trying to localize itself [23]. While this process is on going, there can be times that the robot will not be in a position from which can determine a single state from others. That's when aliasing happens in robots. This aliasing phenomena is called perceptual aliasing and describes the hidden states of the environment [7]. The mapping between states of the world and sensations of the agent is not one-to-one. If perceptual limitations allow the agent to perceive only a portion of its world, then many different world states can produce in the same percept. Also, if the agent has an active perceptual system, meaning that it can redirect its sensors to different parts of its surroundings then the reverse will also be true—many different percepts can result from the same world state.

Perceptual aliasing situation of a double meaning. From the good meaning, perceptual aliasing is useful, since we can represent equivalent states with the same action. [7]. From the bad meaning. with perceptual aliasing we can confuse these states, since the required actions are different [7]. Thus, with perceptual aliasing we can have generalization for the states, but we can sometimes over-generalize it. To avoid this over-generalization

we have to do a selection to the hidden states we want to remove them and uncover the correct states that impedes task performance. The techniques that identify the states usually use history information to uncover the hidden state [31].

Chapter 3

Related Work

The methods and the algorithms, which have been proposed in order to solve and encounter problems in the first steps of DTP were not efficient in the terms of time and space complexity, and they are thus difficult to use in realistic environments. This raised the need of approximations, based on the primary methods to achieve better convergence rates and optimality results.

In this chapter we will analyze some of the most important RL and DTP approximations in the terms of dynamic programming. We will highlight the methods they used in order to modify and improve the classic algorithms and their complexity advantages.

3.1 Least-Squares Policy Iteration

Least-Squares Policy Iteration (LSPI) [28] was inspired by the Least-Squares Temporal Difference (LSTD) [2] learning algorithm. LSPI is an off-policy learning algorithm for control problems and can use any collection of training data from any sampling distribution. Finally, LSPI is a model-free method, since it does not require to use or learn a model.

LSPI learns $\hat{Q}^{\pi}(s, a; w)$ as an approximation of the real value function $Q^{\pi}(s, a)$, where w are the parameters of the approximation architecture. LSPI focuses on linear architectures, which consist of combinations of k basis functions:

$$\hat{Q}^{\pi}(s, a; w) = \sum_{j=1}^{k} \phi_j(s, a) w_j \tag{3.1}$$

where the w_j are the parameters and $\phi_j(s, a)$ are the basis functions. An important requirement for the $\phi_j(s, a)$ basis functions is that they have to be linearly independent.

The entire approximate value function can be written in a matrix multiplication form:

$$\hat{Q}^{\pi} = \Phi w^{\pi} \tag{3.2}$$

where Φ is a $|A||S| \times k$ matrix and w^{π} is a column vector of length k.

We can recall from Chapter 2 that action-state value function Q^{π} is the fixed point (solution) of the Bellman equation: $Q^{\pi} = R + \gamma P \Pi Q^{\pi}$. Thus, if our approximate state-action value function can be a "close-enough" fixed point to the above equation, then it will be "close-enough" fixed point to the real action-state value function:

$$\hat{Q}^{\pi} = \mathcal{P}\big(R + \gamma P \Pi \hat{Q}^{\pi}\big) \tag{3.3}$$

where $\mathcal{P}$ is the orthogonal projection back to the sub-space of the k-basis functions. Plugging in the approximation of Equation 3.2 leads to a square linear system and solving for w yields this "fixed point" solution. LSPI uses the training data to estimate the matrices of this linear system and iterates over different w, which correspond to improving policies, until convergence.

3.2 Regularized Policy Iteration

In the core of RL and DTP, the procedures of evaluating and updating a policy π are the basis. The goal of these steps is to find or approximate a state V or action-state Q value function. In the context of control, these value functions are not known in advance and the cost of calculating them is high. Regularized policy iteration [24] is an approximate policy-iteration based reinforcement learning algorithm. There are two different regularized PI algorithms, which use two policy evaluation methods. The first one is Bellman residual minimization (BRM) and the second one is least-squares temporal difference (LSTD).

BRM comes from the fixed-point equation for the Bellman operator $Q^{\pi} - T^{\pi}Q^{\pi} = 0$. If we replace the Q^{π} with another function Q and the magnitude of the Bellman residual $\|Q - T^{\pi}Q\|$, is small, then Q function is a good approximation of Q^{π}. Another way to calculate the magnitude of the Bellman residual is with the use of L^2-norm, where it leads to an optimization problem for the function Q. Hence, we define the loss function $L_{BRM}(Q;\pi) = \|Q - T^{\pi}Q\|_v^2$, where v is the stationary distribution of states in the input data. Using samples (X_t, A_t) and by replacing $(T^{\pi}Q)(X_t, A_t)$ with its sample-based approximation $(\hat{T}^{\pi}Q)(Xt, A_t) = R_t + \gamma Q(X_{t+1}, \pi(X_{t+1}))$, the empirical counterpart of $L_{BRM}(Q;\pi)$ can be written as:

$$\hat{L}_{BRM}(Q;\pi,n) = \frac{1}{nM}\sum_{t=1}^{n}[Q(X_t,A_t) - (R_t + \gamma Q(X_{t+1},\pi(X_{t+1})))]^2 \quad (3.4)$$

Unlike BRM that minimizes the distance of Q and $T^\pi Q$, LSTD minimizes the distance of Q and $\Pi T^\pi Q$, the back-projection of the image of Q under the Bellman operator, $T^\pi Q$. Formally, this means that LSTD minimizes the loss function $L_{LSTD}(Q;\pi) = \|Q - \Pi T^\pi Q\|_v^2$. The LSTD solution can therefore be written as the solution of the following optimization problems:

$$h_Q^* = \text{argmin}_{h\in F^M}\left\|h - \hat{T}^\pi Q\right\|_v^2, \hat{Q}_{LSTD} = \text{argmin}_{Q\in F^M}[\left\|Q - h_Q^*\right\|_v^2 \quad (3.5)$$

where the first equation finds the projection, and the second one minimizes the distance of Q and the projection.

Concluding, the creation of a regularized PI algorithm can be done with two ways. As it is clear, the first way to do it is to use the approximate policy evaluation of Bellman residual minimization and the second one the least-squares temporal difference method. These two methods are minimization problems in their basis, and has shown that can work well for the Q approximation, providing optimal results under certain conditions [24].

3.3 Approximate Lambda Policy Iteration

Lambda policy iteration [16] is an approximate dynamic programming algorithm based on policy iteration and on the notion of temporal differences. This method is related with DP problems with discounted and undiscounted cost. The idea behind this approach is that the discount factor λ, can be reduced without updating the value function of a given policy.

The motivation for this method was large-scale problems where other approximations of the value functions become interesting. With the Monte Carlo method we evaluated the policy, and as we can see from the definition of Δ_t, the λ parameter has a dominant role.

$$\Delta_t(s) = \sum_{\pi=0}^{\infty}\mathbb{E}[(\alpha\lambda)^\pi d_t(s_\pi, s_{\pi+1})|s_0 = s], \forall s. \quad (3.6)$$

where this sum over the policies π denote the new calculated policies over time and $\alpha\lambda$ denotes the discounted DP problem, for $\lambda \in [0,1]$.

In cases where policy evaluation step does not perform well, a reasonable approach is to choose a policy π with the best tolerance from the optimum.

Acknowledging this, the rate of the cost function J_t, and the usage of the λ parameter, require more number of policy iteration steps to perform well.

Thus, an appropriate λ-PI method with a linear approximation architecture will look like this:

$$\hat{J}(s,r) = r(0) + \sum_{k=1}^{K} r(k)\phi_k(s), \tag{3.7}$$

where $r(k), k = 0, 1, 2...K$, are the components of the parameter vector r, and ϕ_k are fixed basis functions. The r values helps to approximate the cost function $\hat{J}(i,r)$.

$$\pi_t(i) = \text{argmin}_{u \in S} \sum_{j=0}^{m} p_{s,j}(g(s,u,j) + \hat{J}(j,r_t)), \forall s \tag{3.8}$$

We obtain our M trajectories and the parameter vector r_{t+1} as:

$$r_{t+1} = \text{argmin}_r \sum_{m}^{M} \sum_{k}^{Nm} (\hat{J}(i_{m,k},r) - \hat{J}(i_{m,k},r_t) - \sum_{s=k}^{Nm-1} \lambda^{s-k} d_t(i_{m,s}, i_{m,s+1}))^2 \tag{3.9}$$

Concluding, λ policy iteration can be viewed as the prior method of $TD(\lambda)$. $TD(\lambda)$ uses gradient-like methods, while λ policy iteration linear algebra packages [16].

3.4 The Alternating Policy Iteration Method

In the context of solar tracking systems, in 2015 Panagopoulos, Chalkiadakis and Jennings [17] created an approximate dynamic programming method to attack the solar tracking problem. Inspired by policy iteration they created an approximation approach where they solve alternately the parts of the decomposed space.

In the context of approximations and optimization the alternating method is not new, it is used in problems where the need of dimensionality reduction or decomposition is necessary. In the field of decision-theoretic planning and reinforcement learning this approach was the first to be implemented. Thus, it is worth studying and extending.

3.4.1 Dimensions Decomposition

The main and interesting part of this approach is the decomposition process in their environment. Their environment was a photovoltaic system with two dimensions of freedom(Azimuth and Slope) and the goal, for this case,

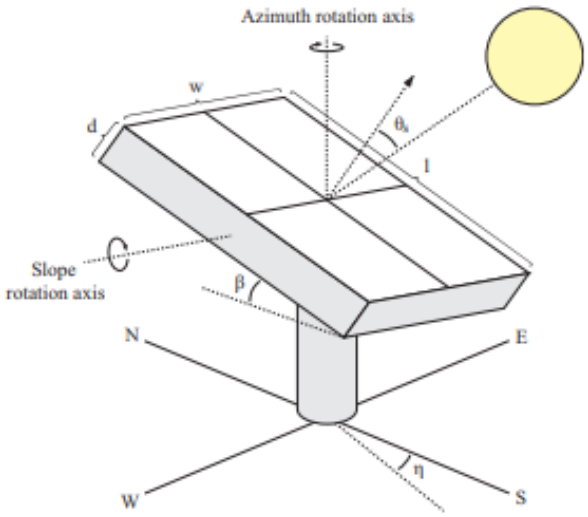

Figure 3.1: Dimension decomposition in a dual-axis photovoltaic system [17]

is to move each dimension in the right position in order to obtain the maximum energy from the sun. In Figure 3.1 we can see an example of a dual axis photovoltaic system, where the two dimensions can be handled independently. This alternating policy iteration (AltPI) method they used has many applications in every static environment and in all kind of decision making problems.

Therefore, in our thesis we note that AltPI can be applied in the most general environment, the grid-world. A grid-world can simulate many environmental cases, since the use of obstacles (*walls*) in it or the different combinations of possible actions create many different scenarios of environments. The decomposition process in a grid-world will split the two dimensions into two vectors, which we call columns and rows. With the combination of these vectors, we will get the global grid again. As we can see in the Figure 3.2, we have the two vectors, the row and the column, and the global grid which can be produced from the two vectors. We can represent every a_s square in the grid with the combination of a_{ci} and a_{ri} respectively and vice versa. An advantage of this method, is that we can represent the same information with fewer states. As it is obvious, the row and column states are eight (four in each vector) in comparison with the global grid, which are sixteen.

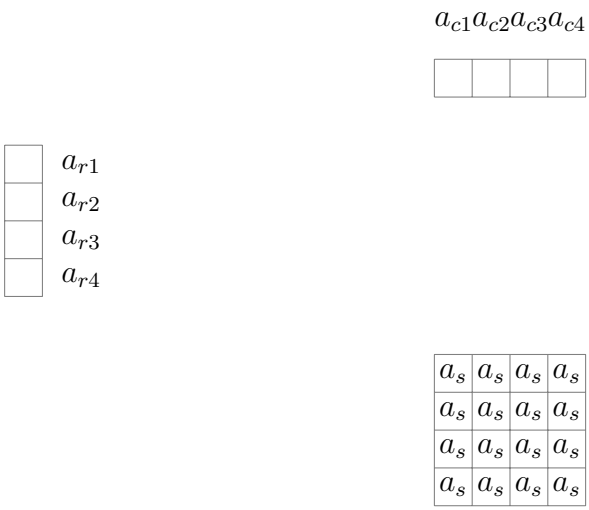

Figure 3.2: Decomposition in an empty grid-world

3.4.2 Alternating Method

Having introduce the decomposition process, now we can proceed to describe the alternating method [17], re-defined here in the context of grid worlds, which every decomposed dimension can converge independently to their policies and produce a global result from their combination.

More specifically, the alternating method contain an initialization of a single policy π_i, where i represents each of the decomposed dimension(in the grid-world case we have row policy and column policy), for every decomposed state space. After this initialization, for every $\pi_k, where\ k \in i$, the policy iteration procedure starts by holding all the π_{i-k} policies stable. The output of that algorithm, π_k, is then fed in a second policy iteration algorithm to compute another $\pi_{k'}$ policy. By the time of these alternating iteration and at the end of them, all these policies π_i, are combined to produce the global policy of the whole state space, π. In algorithm 5, we present the overall technique, where all these policies π_i are alternating run until convergence.

Algorithm 5 Alternating Policy Iteration

procedure ALT PI(π)
initialize $V(s) = 0 \quad \forall s \in S$ ▷ Initialization
initialize $\pi_{row},\ \pi_{col}$ *based on* π

while π_{row} *and* π_{col} *is not stable* **do** ▷ Main procedure
$\pi_{col} =$ *policy iteration on column*$(\pi_{col}, \pi_{row}, \pi)$
$\pi_{row} =$ *policy iteration on row*$(\pi_{col}, \pi_{row}, \pi)$
$\pi' \leftarrow Comb(\langle \pi_{col}, \pi_{row} \rangle) =>$ see Alg. 7

return π'

The highlight of this algorithm is the PI procedure in each dimension. In the context of this thesis we created a grid world, which we will briefly discuss in the next chapters, so the decomposed dimensions are columns and rows. In Algorithm 6, we describe the policy iteration procedure for the dimension of rows, since the procedure for columns is the same. As we see we combine every row state with the column states to create the global states of our grid. After that we combine the actions, by checking of course the validity of the combination given the conditions of the grid, doing policy evaluation and policy improvement steps iteratively, until the row policy stabilize (and the column policy).

The procedure for each dimension together with the combination of actions method is described below:

We would like to discuss the combination of actions method (algorithm 7). As we can see in the pseudocode we have an $if - else$ state, which calculates the global action from the decomposed actions. If the combination of π_{row} and π_{col} is invalid, then we need to search for a new global action, which will not include the result of the calculated decomposed actions. In the environment of the solar tracking systems there was no validity problem of action combination. The reason we present this method, is to point out that in case we can not combine these actions (validity problem for some reasons) we have to find a new action, feasible for the current state of the grid-world.

Algorithm 6 Policy Iteration in each dimension

procedure PI FOR ROWS($\pi_{row}, \pi_{col}, \pi$)

while π_{row} *is not stable* **do**

for each $s_r \in S_{row}$ **do** ▷ Policy Evaluation Step
for each $s_c \in S_{col}$ *based on* π_{col} *and can be combined with* s_r **do**

$s \leftarrow \langle s_r, s_c \rangle$
$a \leftarrow \pi(s)$
$V(s) \leftarrow \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$

for each $s_r \in S_{row}$ **do** ▷ Policy Improvement Step
for each $s_c \in S_{col}$ *based on* π_{col} *and can be combined with* s_r **do**

$s \leftarrow \langle s_r, s_c \rangle$
$\pi_{row}(s_r) \leftarrow \text{argmax}_{a_{row}} \sum_{s'} P(s, a, s')(R_a(s, s') + \gamma V(s'))$,
where $a = Comb(\langle a_{col} = \pi_{col}(s_c), a_{row} \rangle)$ => see Alg. 7

return π_{row}

Algorithm 7 Combination of actions(original method)

procedure COMB(a_{row}, a_{col})

$\alpha' \leftarrow \langle a_{col}, a_{row} \rangle$
if α' *is valid action* **then**
leave α' *unchanged*
else
Improvement Step
find α' *which is* $feasible$

return α'

Chapter 4

A Novel Alternating Policy Iteration Algorithm

In the previous chapter, we have described the AltPI method [17]. In this chapter we will introduce a novel method based on AltPI, which encounters the phenomena of information aliasing and performs promisingly better from the classic approaches. These phenomena appeared in our problem from the decomposition process we applied in our environment. Also, we will describe the first steps we tried that finally led us to the novel method.

4.1 Grid World

Because of the nature of our algorithms, we needed an environment that will be generic and static. Also this environment should have the ability to scale its state space dimensions, for more stable and accurate results.

Thus, we created a grid-world with NxN dimensions, where N determines the column and the row number of states. As we have mentioned in a previous chapter, in the grid-world we will use some kind of obstacles. These obstacles will be referred to as $walls$, and the purpose of them is to not let the methods to create a straightforward policy, but instead a more complex policy π. To understand better the meaning of these $walls$, imagine a policy π in a maze full of them. It would be an environment with the maximum difficulties for the algorithms. These obstacles will also create aliasing phenomena in the grid-world, which led us in our novel method. The more the $walls$ the more intense information aliasing phenomena will be in our environment.

Moreover, the action space A was defined in order to cover all directions in grid-world. More specifically the actions space is defined below:

$$A = \{right, left, down, up, up - right, \\ up - left, down - right, down - left, stay\} \tag{4.1}$$

This flexibility in actions, provides a better handling in the case of alternating policy iteration. In this case, due to decomposition, we have to combine actions from column space with actions from row space. More specifically:

$$A_{rows} = \{up, down, stay\} \\ A_{col} = \{right, left, stay\} \tag{4.2}$$

We can see, that the straight forward, combination of the A_{rows} and A_{col} action spaces, gives us as result the A space.

We define the rest dynamics of our Markov decision process, transition probability table and reward table, appropriately. For the reward table, in each grid-world we initialize, we have one coordinate that we pointed as goal. This coordination could be anywhere, but in the maze should be the state for the exit (to create an exit policy from it). So for every state, where the taken action led to the goal, we have a reward. Furthermore, for the transition probability table we tried to initialize it as good as we could in order to run well the algorithms. Moreover, if we are in state s_0, and we choose action a_{right}, then the probability that will take us to the next states is described below:

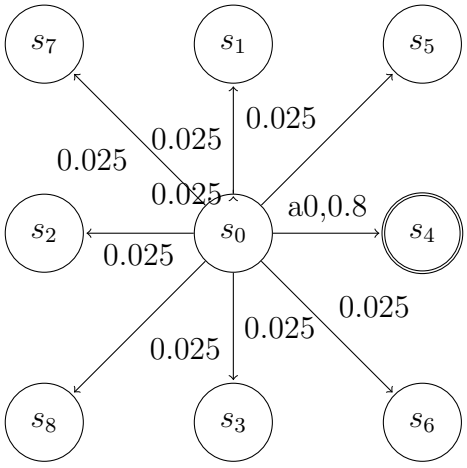

Figure 4.1: Example of transition probability table for our MDP

It is noticeable, that with probability 0.8 we go to the correct state, in this case from s_0 to s_4. The rest of the remaining probability, 0.2, is equally

distributed to the rest of its neighbours (0.025, since the rest neighbors are 8). With these probabilities, we tried to create a consistent environment, to make the actions more safe. In any case, this probability can easily change, to create a more inconsistent environment for further results.

Having described how the initialization of our MDP is done, and what type of environment we used, it would be a proper moment to talk about the obstacles *walls* and their meaning.

Firstly, there are three types of grid-world that we created and tested our algorithms. The first one is the simplest one, where there are no walls in the grid-world. In the second case, we initialize a number of walls (the number of them was analogous to the grid size, but they were not many) with random coordinates, but not the same with the goal. These *walls* was placed with a specific order to not block a state from its way to goal (i.e. the goal it could be reached from every state). For the last and more complex case we created a maze. which contains the maximum number of *walls* in the grid-world. To create a maze we used a recursive algorithm, which is based on depth-first search algorithm [29].

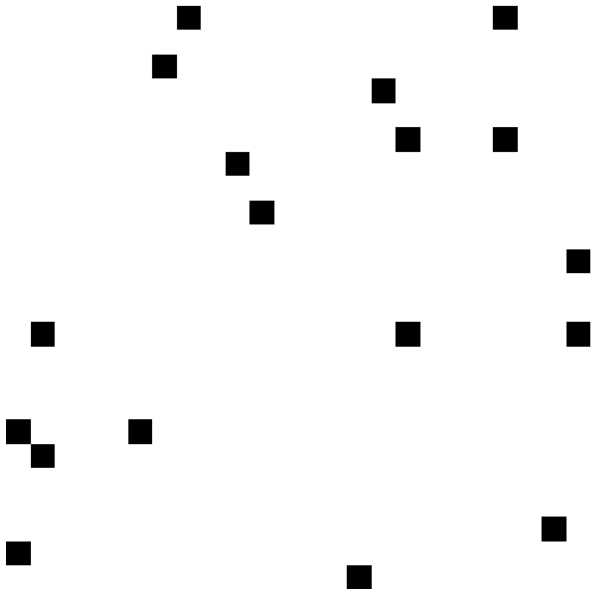

Figure 4.2: 25x25 Grid-World with random walls

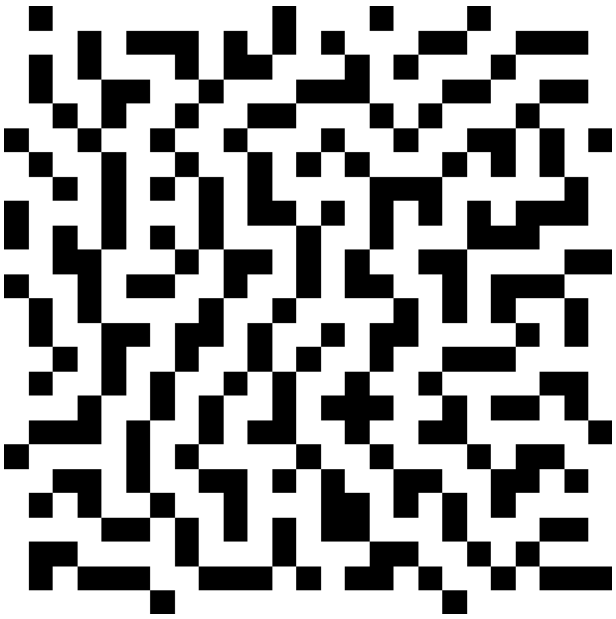

Figure 4.3: 25x25 Maze Grid-World

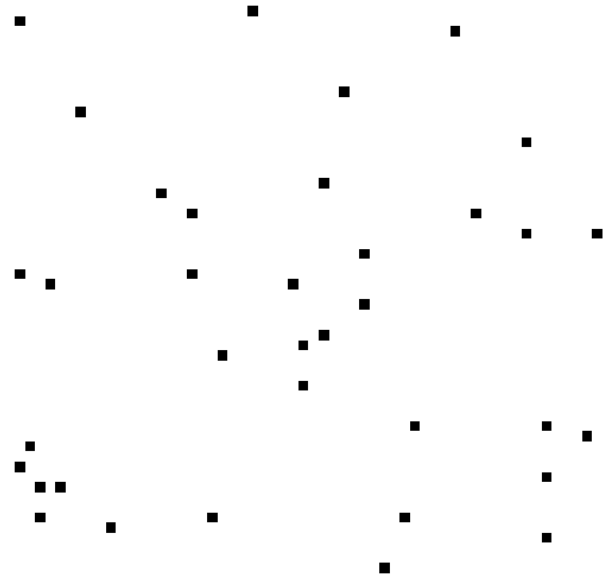

Figure 4.4: 60x60 Grid-World with random walls

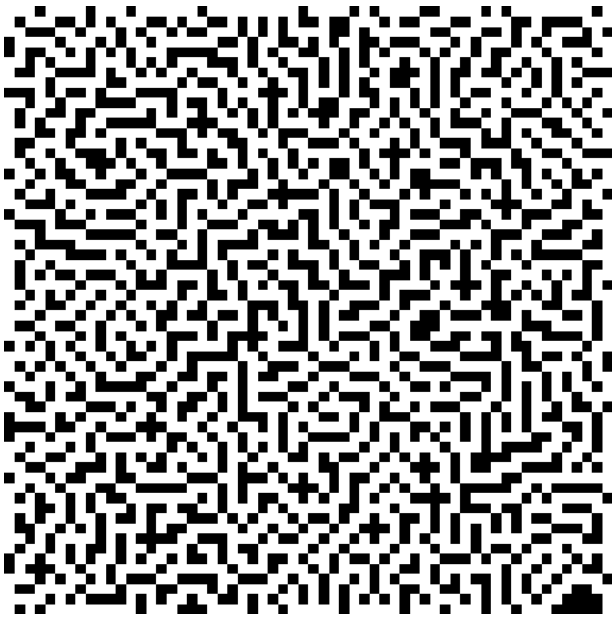

Figure 4.5: 60x60 Maze Grid-World

In Figures 4.3, 4.2, 4.5, 4.4, we see an example of two grid-world cases with random walls ($1 - 5\% walls$ of the total space) and two maze grid-world cases ($38 - 42\% walls$ of the total space). For the simplest case of an empty grid-world, we have an example of it in Figure 1.1.

The reason we choose these type of environments is not random. Information aliasing is a well defined phenomena, which appears when dimensionality reduction or decomposition happens.

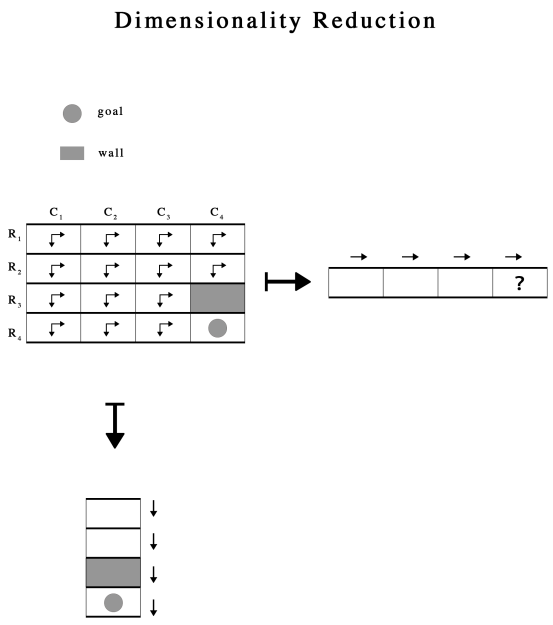

Figure 4.6: Grid-world decomposition with walls

As we can see in Figure 4.6, we have an example of decomposition process in a grid-world with *walls*. When we decompose the whole grid into row vector and column vector, we notice that the information of *walls* can not be passed in the vectors. As a result we lose the *wall* information in the lower dimensions, and that creates a problem in the combination of actions. Thus, if one state of the row vector, has the information of wall, we lose the action of that state and consequently we lose optimality in the global policy. In the action spaces we see in Eq. 4.2, only the combination of them can give us the global action space of Eq. 4.1.

In addition, it is clear that an optimal global policy π, will be provided by optimal decomposed policies, π_{row} and π_{col}. In addition, the information of goal states and *wall* states are reachable from the composition of row and column vector. So, the approximate alternating algorithm, can find optimal decomposed policies, π_{row} and π_{col}, which subsequently will provide optimal global policy π. The challenge of this algorithm would be to find optimal global policy π, even when the decomposed policies, π_{row} and π_{col}, are sub-optimal. This case is the one we face with the information aliasing, since the *walls* do not allow the combination of column and row policies.

4.2 Principal Component Analysis

In the previous section we defined our environment, our MDP and the information aliasing phenomena, which appear through decomposition process in a grid-world. To tackle aliasing phenomena the first approach we took was to focus on value function approximation. The idea behind this approximation was to obtain a prior knowledge from a converged value function (produced by PI or VI methods) and use it in the initialized value function of alternating policy iteration algorithm. This value function would contain the proper information of the grid-world i.e. the hidden states (the *walls* in the grid), which is lost from decomposition process and help the algorithm to converge in an optimal (or sub-optimal) value function. With this value function the algorithm should be in position to tackle information aliasing and produce a proper policy π for the grid-world.

There are many ways to extract information from a value function and many ways to approximate a new one. For our case the most suitable method is principal component analysis (PCA). Principal component analysis [10], is a statistical procedure that uses an orthogonal transformation to convert to a set of possibly correlated variables into a set of values of linearly uncorrelated variables called principal component. PCA can be done by eigenvalue decomposition of a data covariance matrix or singular value decomposition [33] [20] of a data matrix, usually after a normalization step of the initial data. The results of a PCA are usually discussed in terms of component scores, sometimes called factored scores. The weight of these scores, present the correlation of the data which perform the analysis.

In the grid-world, we performed PCA in the column and row vectors as a pair, to see and extract any kind of correlation, which could appear between them. This correlation would mean that the vectors "hide" information, which appears in the global grid, and subsequently leads to *wall* information, which create the aliasing phenomena. The information of correlation between them would be extracted from the first principal component of the PCA and would be used as a prior knowledge to the value function of alternating policy iteration method.

After applying the procedure of PCA in our vectors, we calculate the component score of the first principal component. In Figures 4.7, 4.8 and 4.9 we can see that the outcome variables of PCA coincide with the first principal component.

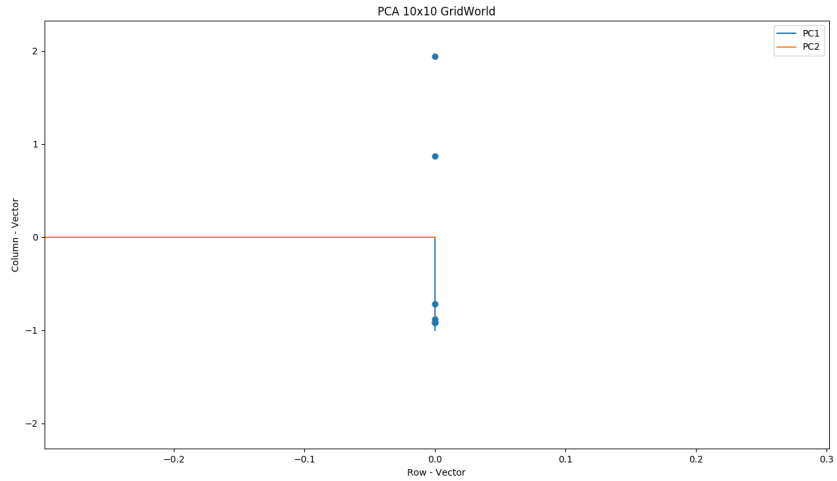

Figure 4.7: PCA analysis in 10x10 grid-world

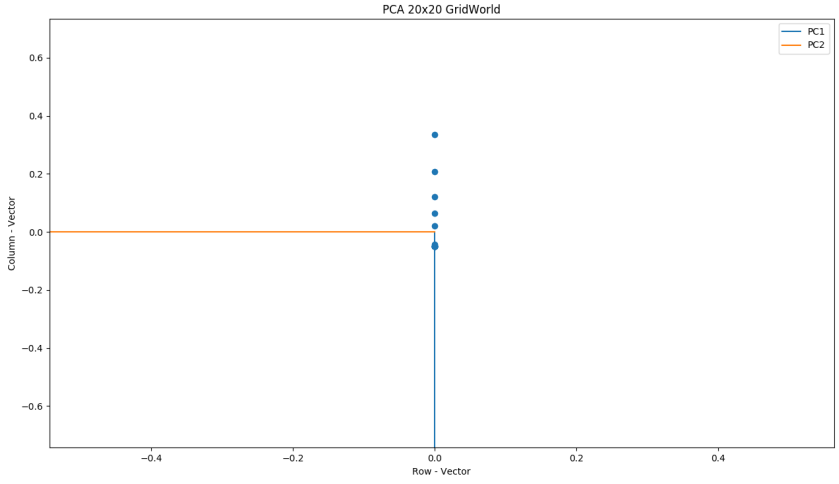

Figure 4.8: PCA analysis in 20x20 grid-world

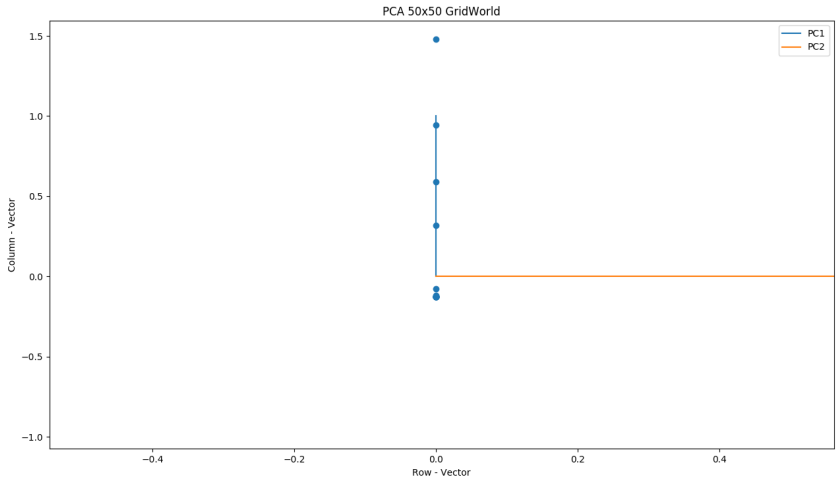

Figure 4.9: PCA analysis in 50x50 grid-world

These results end up with a component score of zero, subsequently we do not get any correlation between row and column vectors. The interesting part here is that the converged value function did not gave us the correlation we expected from the hidden *walls* and that led us to new approaches in order to encounter information aliasing. These approaches focus on policies improvement steps and ways that concern decision theory, based on value functions.

4.3 A novel approach

After changing the whole approach of the problem and focusing in ways of policy decision making we ended up with a novel method. This method is based on alternating policy iteration algorithm, but since this method may create information aliasing phenomena in some environmental cases, we improved the part of the algorithm which lacks and could not tackle aliasing.

In Chapter 3 we talked about the ways that AltPI works and calculates a value function and a global policy. Specifically, in the algorithmic method, which calculates a global action from the two separate actions (row and column action) is where the aliasing phenomena affects the algorithm. The *walls* do not appear in the decomposed dimensions and hence, when we combine two (valid for the decomposed space) actions the new one has chances to not be feasible. This means that the produced action will lead to a *wall*-state and thus we can not use it for a future value function

calculation.

One of the practical problems that aliasing creates is that we cannot find a way to calculate a global action, in case that the produced action is invalid. The reasons that the AltPI could not calculate another proper global action are two. The first one has to do with the nature of the algorithm, since to alternate between the decomposed space it has to keep the rest of the space frozen and hence there is only one dimension of freedom in the action selection. The second reason is a combination of the first reason and the decomposition process. Because the algorithm works in lower dimensions and drops the information of the *walls* and since it keeps only one dimension of freedom, this dimension can not decide by itself what local action to choose in order to be the best global action for the grid-world. These observations helped us to provide a method, which can handle the aliasing phenomena.

To introduce our method we have to consider some things. Firstly, we need to know the original goal of the decision making problems, which is to calculate a value function v_π and produce via this function a greedy method the global policy π. This global policy will help to calculate a new value function, in turn, until this procedure converge. The AltPI method decomposes the state-action space in order to make the above procedure faster, i.e. to find a value function and a global policy. The question here is if we can use the whole state-action space in the states that we have information aliasing phenomena in order to counteract it. In other words, when the algorithm cannot combine the two separate action because of the *wall*, we will choose an action from the global space where the information of it is known.

In algorithm 8, we can see the pseudocode of action combination with the improvement step that counteract information aliasing.

Algorithm 8 Combination of actions (Improved)

procedure COMB(a_{row}, a_{col})

$\alpha^{'} \leftarrow \langle a_{col}, a_{row} \rangle$
if $\alpha^{'}$ *is valid action* **then**
leave $\alpha^{'}$ *unchanged*
else▷ This is a policy improvement step for the information aliasing case
$\alpha^{'} \leftarrow \text{argmax}_{a \in A} \sum_{s'} P(s, a, s^{'})(R_a(s, s^{'}) + \gamma V(s^{'}))$,
▷ With this policy improvement step we calculate a global action, which is different from the combination of a_{row} and a_{col}
return $\alpha^{'}$

In the above method observe the case where the two decomposed action

cannot be combined because of the aliasing phenomena. In that case, we use the whole space to calculate a global action, ignoring the two decomposed actions.

To understand better this method, we need to mention that because of the nature of the environment (static), all the dynamics of the MDP are known. Consequently, at any time we have full access in the sets of state space S and action space A. The idea that algorithm 8 was based is the following: In the combination process of decomposed actions a_{row} and a_{col} we face a problem when the result, a, is invalid, i.e. information aliasing. At this part, because we know exactly when the actions are valid, we thought that if and only if the combined action is invalid, then we face $walls$. So, a solution to this problem would be to to run a complete policy improvement step for that state. This extra step will illustrate us the information of *wall* and will let us decide, based on value function, a global action which avoids the *wall*. We named this method aliasing-aware alternating policy iteration (AAAPI), since it focuses and counteracts information aliasing phenomena.

This procedure will provide our a solution with global policies π that follow the value function, but the decomposed policies, a_{row} and a_{col}, will be in a state of sub-optimality, since the combination of them will not give us the global policy. Indeed, the goal of the problem is to obtain a global value function and a global policy. A drawback of this algorithm is that adds extra complexity to the action calculation. In the next chapter we will see how this method compares with the classic decision making algorithms.

4.4 Discussion

In this chapter we have seen the steps of this thesis and how they evolved. We saw the first part, which was the initialization and creation of our MDP environment. Thereafter, we continued with our first approach that would led us to the solution of the information aliasing problem. Finally, we introduced a novel method AAAPI based on a DTP algorithm AltPI, which counters information aliasing phenomena and solves our MDP.

Chapter 5

Experimental Evaluation

After the creation of a novel algorithm in the context of DTP, which can encounter the information aliasing phenomena, the next step is to evaluate it against two classic DP algorithms and AltPI. The evaluation will be in two contexts. Firstly, we will compare these algorithms in their computational time and secondly to their optimality.

5.1 Experimental Setup

For the needs of this thesis, to experimental evaluate a DP algorithm like AAAPI and AltPI, we had to implemented some classical DP algorithms, PI and VI, proper for the problem (DTP). Firstly, we setup the environment, as we have described in Chapter 4, creating three different grid-worlds in order to cover the most of the cases. These cases has to be the same for our algorithms, otherwise the results would not be valid and correct. The first algorithm we implemented was PI, which is the most common DP algorithm for DTP and the one that alternating PI was based. Subsequently, the second DP algorithm we implemented was VI, which follows the same principals with PI, such as policy evaluation and policy improvement. These two algorithms (analyzed in Chapter 2) provide a baseline to solve MDP, in the context of DTP with DP.

Continuing, the next method we implemented is AltPI. In the previous chapter, we have discussed the information aliasing phenomena, which appears in decomposition process, and how they can effect the original alternating PI method. This algorithm will not perform optimally in environments with obstacles. The reason is, as we have referred, that the decomposed policies, π_{row} and π_{col}, will not produce a global optimal policy π, since the *walls* will "block" the combined action. Despite this, our AAAPI variant of AltPI is more general and can perform well in every static environment with aliasing phenomena.

To this end, we have implemented the algorithms and the methods we

used for the need of our thesis. The final step is to run these algorithms to produce their convergence rates, compare them under the same environmental conditions and check their optimality.

5.2 Experiments and Results

In this section, after having built our environment and implemented our methods, we will discuss and present our results.

Firstly, we would like to mention that the results we will present, are under the terms of computational time in a state growing environment. In other words, for a state space S, which is getting bigger and bigger, we calculate the computational time of our methods. The results will be experimental and not mathematically formalized, but since there are bounds for the time complexity of PI and VI, we can evaluate our results under these bounds.

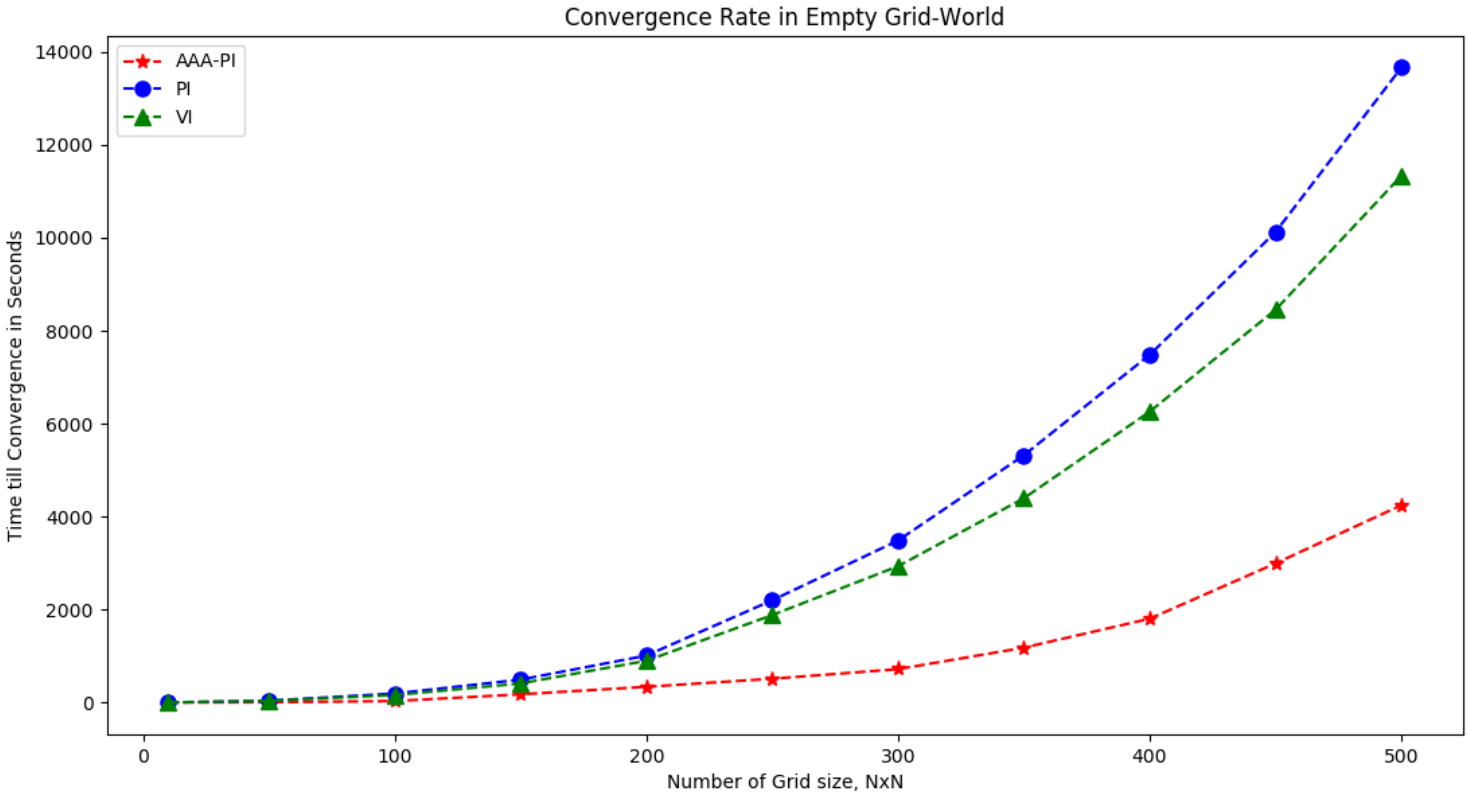

Figure 5.1: Time convergence rate in grid-world without walls

In Figures 5.1 and 5.2, we have the time convergence of the algorithms

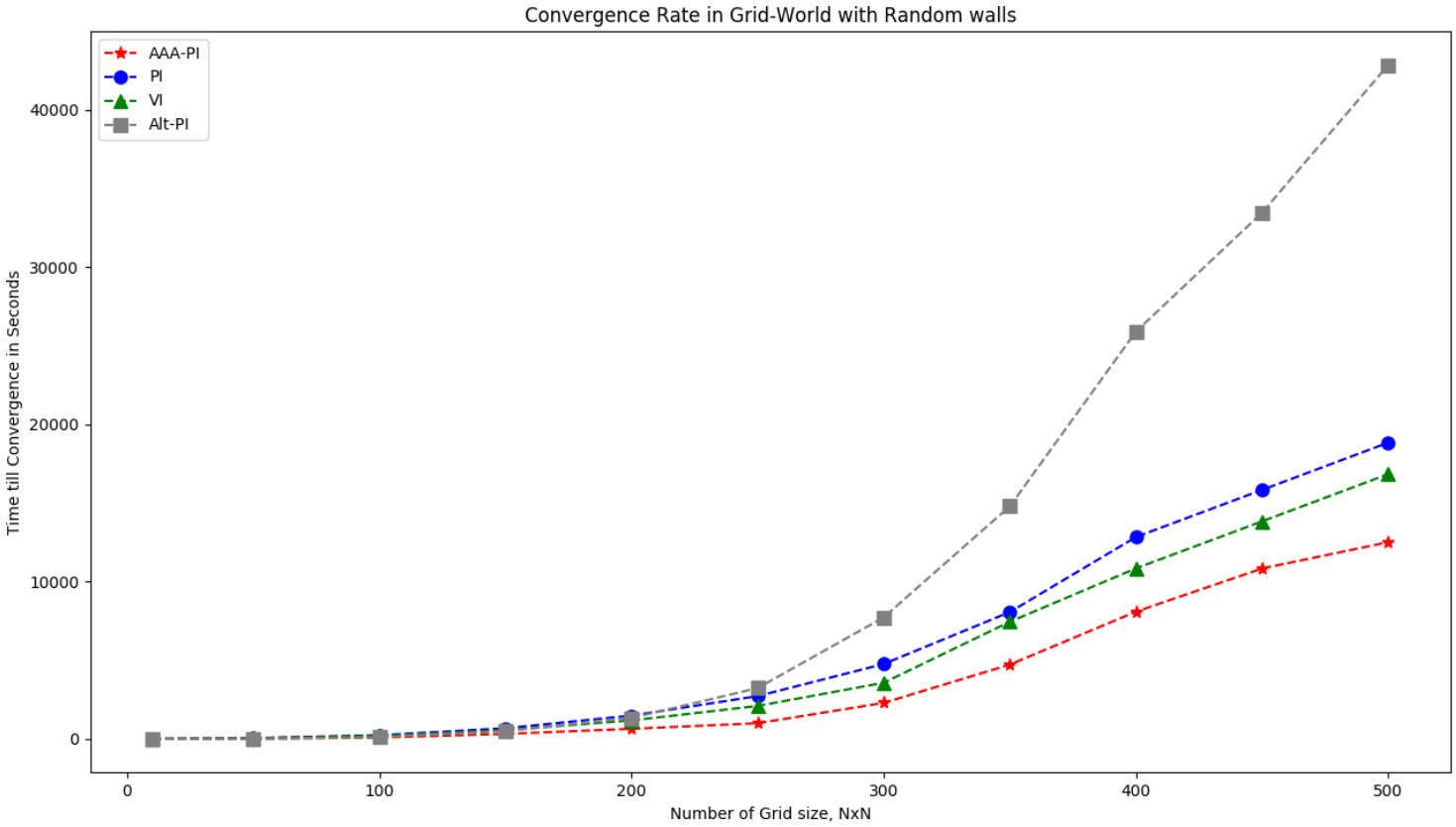

Figure 5.2: Time convergence rate in grid-world with random walls

in two different types of grid-worlds. In the first one, we run the algorithms in an empty grid, where no *walls* appear, so the AltPI will run exactly as it was proposed, and hence will be the same with the AAAPI. This means that the combination of actions can happen normally, because there will be no information aliasing phenomena in the decomposition process. In the second one, we run a grid-world with *walls* in random coordinates analogous to the state space S. In this situation, information aliasing appears and the new combination method (algorithm 8) is necessary to converge the AAAPI algorithm in the right policy.

Concerning the Figures, we notice that the required time, until the algorithms converge, follows a polynomial form. For the two classic algorithms, PI and VI, the time convergence seems to be similar, with a little difference of course, since their iterative way of solving policy evaluation step and policy improvement step, in VI, is more efficient. AltPI is the same with the AAAPI in the first environmental case, since there are no *walls* in the grid-world and hence no aliasing phenomena. Their performance in this environment is better that the two classic methods, as a result of the alternating technique they use. For the second environmental case with the aliasing phenomena, we notice that AAAPI outperforms all three algorithms. This AAAPI method for solving MDP, by decomposing the state space and action space, converges faster, even when information aliasing phenomena takes place and we need the extra steps of policy improvement to calculate for a global policy π. The AltPI does not perform that well in this case, since it struggles with the aliasing phenomena to find a global policy and is even worse than PI and VI. We have to mention that in these two cases, information aliasing was not so intense, in the first case we did not have these phenomena, in the second case the number of *walls* was the $1-5\%$ and the number of *walls* in the maze was around 40%, of the total state space size S. Hence, in the decomposed space the lost information of the *walls* did not effect our new method.

Furthermore, except from the computational time of the algorithms, we considered that the number of iteration, which these algorithms take until they convergence, would be a good metric to study and explain. In Figure 5.3, we output the number of iterations for these four algorithms in a grid-world similar to that in Figure 5.2, i.e with random *walls* in it. The number of iterations has immediate relation with the computational time of these methods, since the polynomial theoretical upper bound depends on the number of steps until the methods calculate the policy. As we can see, PI and VI methods take the same iterations until they converge, while AAAPI uses almost the half of them. The reason of this result, has to do with dimensions of action and state space. In AAAPI the global result comes from the combination of lower dimension, and hence it is reasonable to have less number of iteration in this case. The AltPI uses

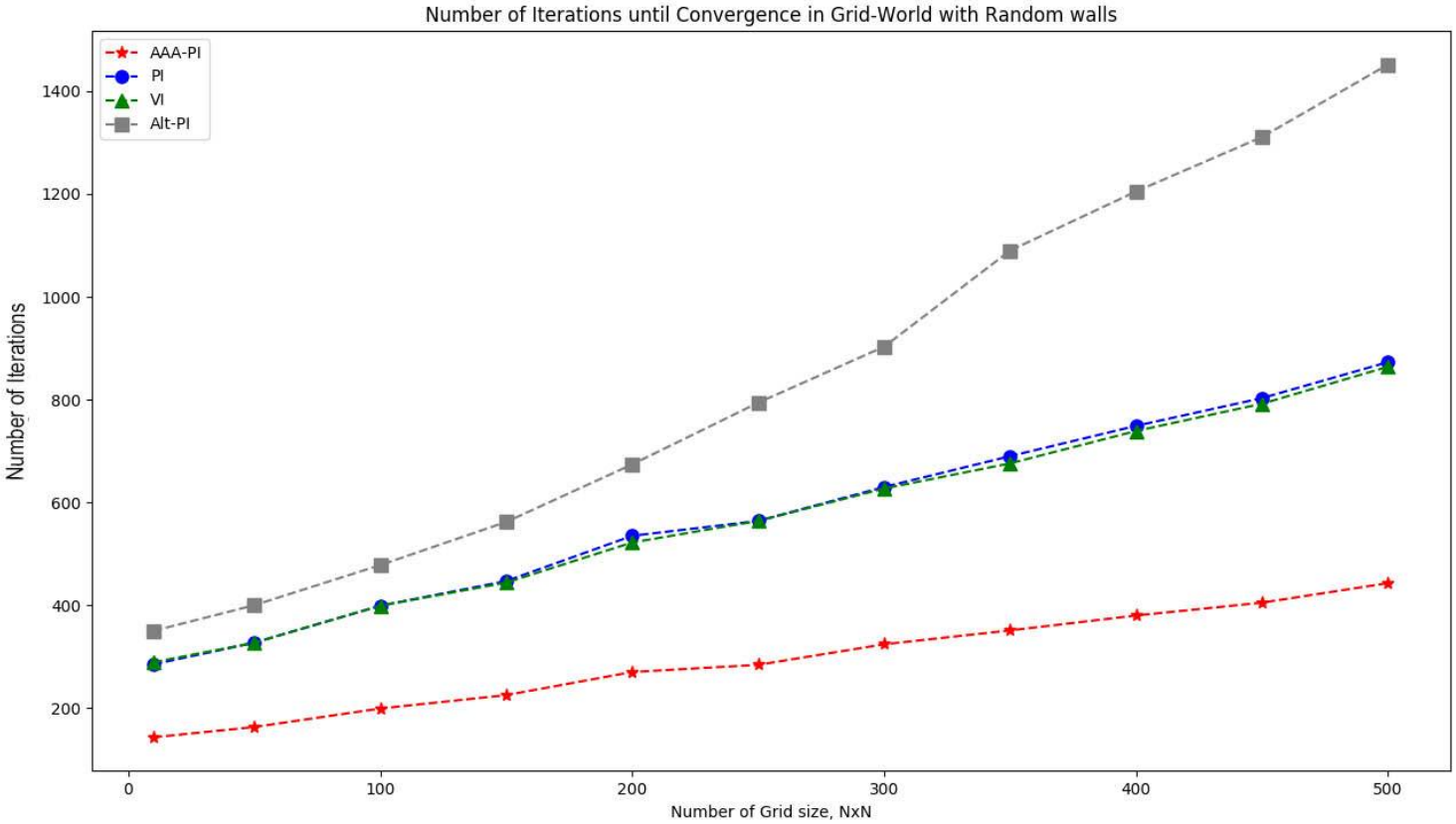

Figure 5.3: Iterations until convergence in grid-world with random walls

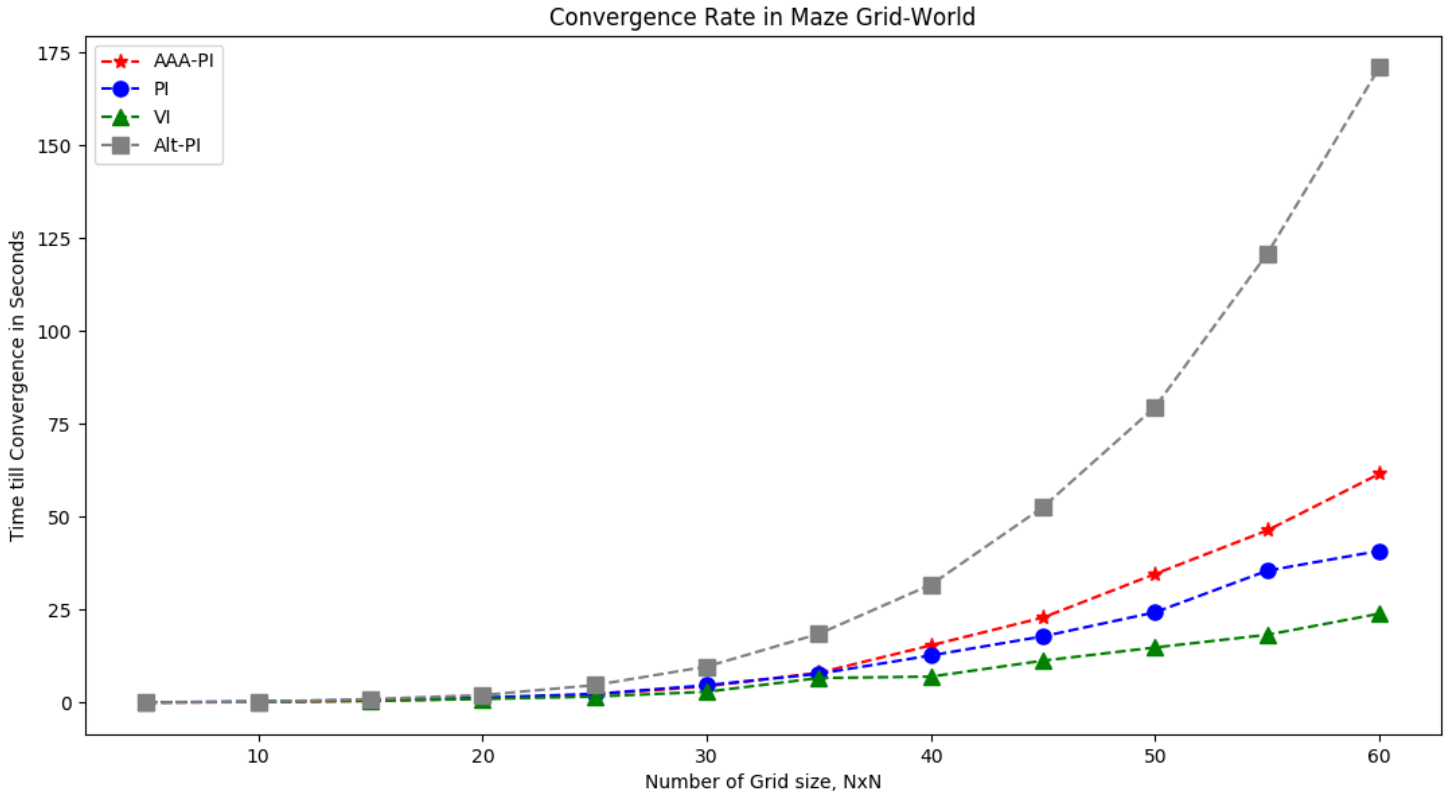

Figure 5.4: Time convergence rate in maze grid-world

the most iterations in comparison with all three methods, since the aliasing phenomena do not allow it to combine the decomposed policies, and thus it cannot come to a decision for the global policy. This metric, also indicates the computational time of the methods in these two environmental cases.

For our final experiment, we used as an environment a maze grid-world. Mazes create paths from states to the goal inside a grid, and the reason we use this type of grid-world is maximize the number of *walls*, inside the grid, and still maintain a path from the states to the goal. A grid-world, which contain many *walls* has a huge impact in the decomposition process, since the row vector and the column vector will loose a lot of information. In this environment, information aliasing phenomenon reach the maximum level and the performance of AAAPI is expected. As we can see in Figure 5.4 (the results in a maze environment), PI and VI time performance follows the same polynomial form, as we saw in the previous cases. The interesting thing here is the computational time of the AAAPI method, since it does not perform so well having the third performance of the four algorithms. The reason of this performance is information aliasing phenomena, which do not help our algorithm to use the advantage of decomposition process, but instead obliges it to use, many times, the extra policy improvement steps. The first expectation here would be that the AAAPI should have the same computational time with PI, since the policy improvement steps are the same. A reasonable explanation that AAAPI doesn't perform the same as PI, would be the dimension freezing and the low degrees of freedom (one dimension at each process) that AAAPI uses. Thus it is natural, in environments were information aliasing phenomena are intense, the AAAPI to not perform well, since the advantage of decomposition and the process in lower dimensions can't exploit. Finally, the AltPI continues to not perform well in environments with aliasing phenomena, thing that is expected since there is no way to counteract the problem of the action infeasibility.

Concluding, all these four methods have been evaluated in terms of optimality with the backward induction method [1], in every environmental case and for every state space S. The three of the four algorithms, PI, VI and AAAPI performed optimally to the goal achievement, as we can see in Figure 5.5 all the actions lead to the goal state. The fourth and final algorithm, AltPI, because of the lack of action combination in environments with aliasing phenomena, could not find the proper global policy and hence could not reach the goal from every state. In Figure 5.6 we have an example of a not optimal policy, since there are states that if we follow the action we will not reach the goal.

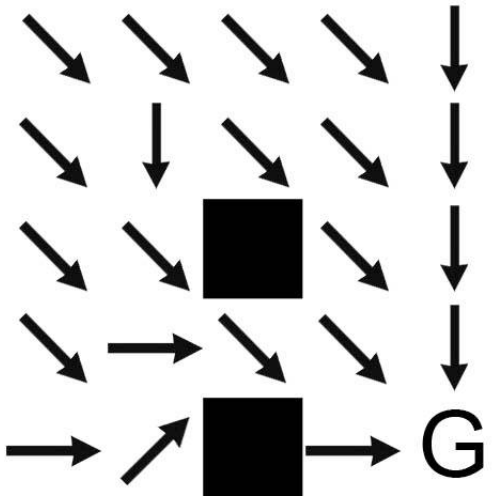

Figure 5.5: Optimal policy in a 5x5 Grid-World

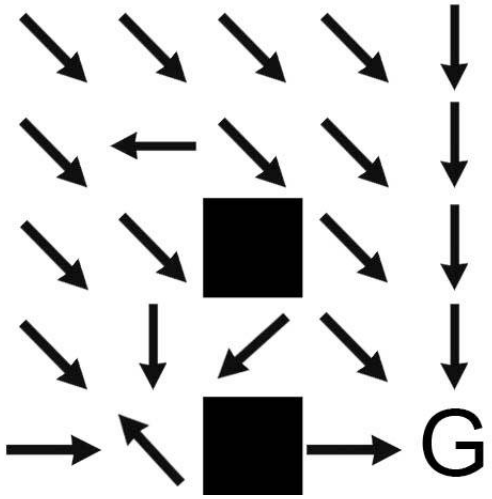

Figure 5.6: Not optimal policy in a 5x5 Grid-World

5.3 Discussion

To summarize, in this chapter we have experimentally evaluated the computational time of four DP algorithms in the context of DTP. We implemented all the needed algorithms in order to have an accurate evaluation, both in computational time and in policy optimality. The creation of proper environments, helped us have a better view of our novel method and learn its advantages and limitations. As we have seen, the novel AAAPI algorithm should be applied in environments without, or with some, information aliasing phenomena. But, in cases like the maze where these aliasing phenomena reach the maximum level, AAAPI should be avoided and simpler classic methods like PI are suggested.

Chapter 6

A Multi-Agent Extension

In this chapter, we will introduce an extension of the AAAPI method in the context of RL. This extension considers a stochastic environment, where more than one agent takes place. These type of environments contain concepts from the game theory field, appropriate for the approximation of a state-action value function q. This value function will be the key for our agent's policy calculation, which has to maximize its utility, depended on other agents polices.

A multi-agent system[37] is a computerized system composed of multiple interacting intelligent agents. These systems can solve problems that are difficult or impossible for an individual agent to solve. Moreover, in game theory, and more specific in competitive zero sum games[26], multi-agent environment are in use. In a zero sum game, the agents are in a situation in which the gain or loss of utility is exactly balanced by the losses or gains of the utility of the other agents. In the context of zero sum games, M.L. Littman proposed a reinforcement learning algorithm (minimax-Q)[38][21], which we will analyse in the next sections, in order to estimate better value functions in the games. Under these conditions, an alternating approach of the minimax-Q algorithm is proposed.

6.1 Matrix Games

In game theory we have a fundamental game, which is the matrix game. Matrix games are defined by a matrix R, where contains the reward of the agents. In these type of games usually the solution (policy of the agent) is computed by minimax theorems [38]. In Figure 6.1, we have a classic example if the matrix game "rock, paper, scissors."

The agent's policy is a probability distribution over actions, $\pi \in Pr(A)$. For "rock, paper, scissors," π is made up of three components: π_{rock}, π_{paper}, and $\pi_{scissors}$. According to the notion of optimality, the optimal minimum expected reward for our agent should be as large as possible. If we could

		Agent		
		rock	paper	scissors
Opponent	rock	0	1	-1
	paper	-1	0	1
	scissors	1	-1	0

Figure 6.1: The matrix game for "rock, paper, scissors" [21]

have a policy that would guarantee an expected score, then we would not care about the opponents policy. Then in the game of π_{rock}, π_{paper}, and $\pi_{scissors}$, the optimal optimal policy π could not be stable.

For π to be optimal, we must identify the largest V for which there is some action of π that makes the matrix game stable. To identify these actions we calculate:

$$V = \max_{\pi \in Pr(A)} \min_{o \in O} \sum_{a \in A} R_{o,a,\pi_a} \tag{6.1}$$

where $\sum_{a \in A} R_{o,a,\pi_a}$ express the expected reward to the agent for using policy π, against the opponent's action o.

As an extension of the matrix games, we can define a general-sum grid game. In this type games we will have two agents (ours and the "opponent"). To define a one-stage general-sum game with 2-players, we will need their action-choice sets A_1 and A_2 and their payoff functions R_1 and R_2. Each payoff function R_i maps an action choice for each of the players to a scalar reward value. For the player i the expected-payoff when players adopt one-stage policies π_1, π_2 is $R_i(\pi_1, \pi_2)$. This is the expected value of R_i weighted by the probabilities under the given one-stage policies [22].

A general-sum grid game consists of a finite set of states S, where each of the states $s \in S$ has its own payoff functions. Moreover, there is a transition table that maps the state-action pair to probability distribution over the next states. The value for a player in a game, given discount factor $\gamma \in (0, 1)$, is the discounted sum of payoffs and is defined as:

$$Q_i(s, a_1, a_2) = R_i(s, a_1, a_2) + \gamma \sum_{s' \in S} T(a_1, a_2, s') Q_i(s', \pi_1, \pi_2) \tag{6.2}$$

where $Q_i(s', \pi_1, \pi_2)$ is the weighted sum of the values of $Q_i(s, a_1, a_2)$. The equation 6.2 represents the value to player i in state s while the players choose actions a_1 and a_2.

There are some cases, in these kind of games, where the opponent player can be defined as a friend or as a foe. In these situations the agents either

cooperate in order to maximize their utility or they compete for their individual profit. In figure 6.2, we can see two cases where the profile of the other player matters, i.e. we need to know whether he is a friend or a foe.

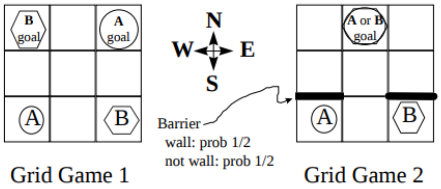

Figure 6.2: The general-sum grid games [22]

It is obvious that in order to reach the goals, the agent A and B, have to develop a policy, which requires the opponents profile, since if the agents "collide" then their action will not be feasible.

6.2 The minimax-Q Algorithm

Given $Q(s, a)$, an agent can maximize its reward using the greedy method as a strategy, choosing the action with highest Q-value. This strategy focuses on the immediate value gain, and it is optimal because the Q-function is accurate for the future rewards. In the zero-sum grid games we define the $V(s)$ to be the expected reward for the policy in the s state, and the $Q(s, a, o)$ as the expected reward for choosing the action a, when the opponent chooses the action o in the state s. Thus, the $Q(s, a, 0)$ can be characterized as the immediate payoff. The $V(s)$ function is the define as the V in equation 6.1 and the $Q(s, a, o)$ as:

$$Q(s, a, o) = R(s, a, o) + \gamma \sum_{s'} T(s, a, o, s')V(s') \tag{6.3}$$

In the Q-learning formulation, an update is performed by an agent whenever a reward r is taken by choosing action a and transitioning from s to s'. This update is $Q(s, a) = r + \gamma V(s')$, which is the one we saw in the Q-learning algorithm. For the zero-sum grid-games we can replace the *max* procedure with the *minimax* generating a new algorithm proper for these type of games.

Algorithm 9 Minimax Q-learning [21]

Initialization :
$Q(s, a, o) = 1$
$V(s) = 1$
$Pr(s, a) = \frac{1}{|A|}$
$alpha = 1$, learning rate
$decay$: controls the rate at which the learning rate decays
$explor$: a probability which controls the exploration

Choose an action :
with probability $explor$, return an action uniformly random
Otherwise, return $Pr(s, a)$ for the current state s

Learn :
After receiving reward r for moving from state s to $s^{'}$
via action a and opponent's action o
$Q(s, a, o) = (1 - alpha)Q(s, a, o) + alpha(r + \gamma V(s^{'}))$
$\pi(s, .) = \text{argmax}_{\pi^{'}(s,.)} \min_{o^{'}} \sum_{a^{'}} \pi(s, a^{'})Q(s, a^{'}, o^{'})$
$V(s) = \min_{o^{'}} \sum_{a^{'}} \pi(s, a^{'})Q(s, a^{'}, o^{'})$
$alpha = alpha * decay$

The algorithm 9, demonstrates the $minimax - Q$ learning algorithm for zero-sum games.

6.3 The Alternating minimax-Q Algorithm

In the previous chapters we have discussed the advantages and the importance of working in lower dimension. In this thesis we illustrated an alternating method, which decomposes the space of the problem and works separately in the decomposed space. This technique is not, always, performing well as we have seen. In environments with intense information aliasing phenomena can be forbidden from use.

In Markov games, and more specific in zero-sum 2-player grid games, like the ones we saw in figure 6.2, the conditions that can develop aliasing phenomena are not that much. We have two cases where the feasibility problem of the decomposed action can exist. The first one appears in the right grid of figure 6.2, where we can have obstacles ($walls$). The second one appears in both grids, where the action of our agent a and the action of the opponent agent o lead them to the same (next) state $s^{'}$. These cases, which create aliasing phenomena seems to be at the same level as the ones we had in the second grid-world, with the random amount of $walls$.

Thus, an alternating method, in the context of reinforcement learning and the zero-sum grid game, with these environmental conditions should be appropriate.

We create a new algorithm based on $minimax - Q$ learning, which uses the alternating method as a way to compute faster the functions $Q(s, a)$ and $V(s)$ and produce optimal policies π for the zero-sum grid games.

Algorithm 10 Aliasing Aware Alternating Minimax Q-learning

Initialization :
$Q(s, a, o) = 1$
$V(s) = 1$
$Pr(s, a) = \frac{1}{|A|}$
$alpha = 1$, learning rate
decay : controls the rate at which the learning rate decays
explor : a probability which controls the exploration

Choose an action :
with probability *explor*, return an action uniformly random
Otherwise, return $Pr(s, a)$ for the current state s

Alternative Learning :
After receiving reward r for moving from state s to $s^{'}$
via action a and opponent's action o

row learning :
for $s_{row} \in S_{row}$ and $a_{row} \in A_{row}$
combine them with the "frozen" column space, to produce global s and a in order to compute Q, π and V
$Q(s, a, o) = (1 - alpha)Q(s, a, o) + alpha(r + \gamma V(s^{'}))$
$\pi(s, .) = \text{argmax}_{\pi'(s,.)} \min_{o'} \sum_{a'} \pi(s, a^{'})Q(s, a^{'}, o^{'})$
$V(s) = \min_{o'} \sum_{a'} \pi(s, a^{'})Q(s, a^{'}, o^{'})$

column learning :
for $s_{col} \in S_{col}$ and $a_{col} \in A_{col}$
combine them with the "frozen" row space, to produce global s and a in order to compute Q, π and V
$Q(s, a, o) = (1 - alpha)Q(s, a, o) + alpha(r + \gamma V(s^{'}))$
$\pi(s, .) = \text{argmax}_{\pi'(s,.)} \min_{o'} \sum_{a'} \pi(s, a^{'})Q(s, a^{'}, o^{'})$
$V(s) = \min_{o'} \sum_{a'} \pi(s, a^{'})Q(s, a^{'}, o^{'})$

$alpha = alpha * decay$

The algorithm 10, illustrates the alternating $minimax - Q$ method. At each separate learning stage we compute a part of the Q, V and π, but at the end of both procedures we have the a full approximation of them. This is another novel algorithm in the field of reinforcement learning, focused on time complexity improvement.

6.4 Discussion

In this chapter, we focused in a different concept of learning with more than one agent. We combined the knowledge of RL methods with the field of zero-sum grid games. The main difference with the classic RL problems, is that we try to approximate our value functions considering the opponents policies with the $minimax$ method. We introduced a very interesting algorithm, the $minimax - Q$, and we created a novel algorithm by combining the alternating method with the $minimax - Q$.

Chapter 7

Conclusions & Future Work

In this thesis, we considered an analysis of decision-theoretic planning methods for MDP solving. In these contexts, we implemented two classic DP algorithms, policy iteration and value iteration, and we created a new algorithm based on the alternating policy iteration method. We faced information aliasing phenomena, which appear in dimensionality reduction or in decomposition process. These phenomena become very intense in some environmental cases, like a maze grid-world, and we managed to encounter them with the new method. Finally, we compared all the implemented algorithms in the terms of time complexity and optimality in different environmental cases, where the aliasing phenomena are more or less intense. The results we produced showed that the new algorithm can outperform the two classic method in the majority of environments, which do not have aliasing or the phenomena are not so intense. In the worst environmental case, which the aliasing phenomena is at the maximum level, our algorithm did not performed so well. This alternating idea can be extended for future work and the next step would be an application in a multi-agent environment, as we have described in chapter 6, as an reinforcement learning method.

Moreover, in computing, a parallel programming model[19] is an abstraction of parallel computer architecture. It contains many parallel models, such as shared memory model [11], message passing [35], implicit interaction [15] etc. In message passing model, parallel processes exchange data through passing messages to one other. These communications can be asynchronous, where a message can be sent before the receiver is ready, or synchronous, where the receiver must be ready. So, in the content of DTP - RL and the optimal MDP solving, we envisage a parallel model of the alternating policy iteration. Each thread of the process working on one dimension and the communication between them should be synchronous, since they would need an update to the decomposed policies in order to continue. With this model, the time complexity should be improved and

give even better results in the cases we saw, such as in environments where information aliasing phenomena are being intense.

Chapter 8

Bibliography

[1] R.J. Aumann. Backward induction and common knowledge of rationality. *Games and Economic Behavior*, 1995.

[2] SJ Bradtke AG Barto. Linear least-squares algorithms for temporal difference learning. *Machine learning, Springer*, 1996.

[3] R. Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 1957.

[4] R. Bellman. Dynamic programming. *Princeton University Press*, 1957.

[5] D.P. Bertsekas. *Dynamic programming and optimal control*. Athena Scientific, Belmont, MA, 1995.

[6] R. Dearden C. Boutilier. Abstraction and approximate decision-theoretic planning. *Journal of Artificial Intelligence Research*, 1997.

[7] L. Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. *AAAI Conference on Artificial Intelligence*, 1992.

[8] R.L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 1986.

[9] CJCH Watkin P. Dayan. Q-Learning. *Machine learning, Springer*, 1992.

[10] S. Wold K. Esbensen P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory*, 1987.

[11] S.V. Adve K. Gharachorloo. Shared memory consistency models: A tutorial. *Computer*, 1996.

[12] C. Boutilier T. Dean S. Hanks. Learning from Delayed Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 1999.

[13] H.V. Hasselt. Double Q-learning. *nips*, 2010.

[14] R. Howard. Dynamic Programming and Markov Processes. *MIT Press, Cambridge, MA*, 1960.

[15] M.P. Jones P. Hudak. Implicit and explicit parallel programming in Haskell. *Disponível por FTP em nebula. systemsz. cs*, 1993.

[16] D.P. Bertsekas S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. *Lab. for Information and Decision Systems Report LIDS*, 1996.

[17] A.A. Panagopoulos G. Chalkiadakis N. R. Jennings. Towards Optimal Solar Tracking: A Dynamic Programming Approach. *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[18] I. Jolliffes. *Principal component analysis*. Springer, 2011.

[19] V. Kumar. *Introduction to parallel computing*. Addison-Wesley Longman Publishing Co., Inc., 2nd edition, 2002.

[20] V. Klema A. Laub. The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, 1980.

[21] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. *Machine learning proceedings*, 1994.

[22] M.L. Littman. Friend-or-Foe Q-learning in General-Sum Games. *ICML*, 2001.

[23] J. Fuentes Pacheco J. Ruiz Ascencio J.M. Rendon Mancha. Visual simultaneous localization and mapping: A survey. *Artificial Intelligence Review*, 2015.

[24] A.M. Farahmand M. Ghavamzadeh C. Szepesvari S. Mannor. Regularized Policy Iteration. *NIPS*, 2009.

[25] L.P. Kaelbling M.L. Littman A.W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence*, 1996.

[26] J. Nash. Non-cooperative games. *Annals of mathematics*, 1951.

[27] C. Guestrin D. Koller R. Parr. Multiagent planning with factored MDPs. *NIPS*, 2002.

[28] M.G. Lagoudakis R. Parr. Least-squares policy iteration. *Journal of machine learning research*, 2003.

[29] S. Beamer K. Asanović D. Patterson. Direction-optimizing breadth-first search. *Scientific Programming*, 2013.

[30] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural networks*, 2015.

[31] D.P. Bertsekas S. Shreve. *Stochastic optimal control: the discrete-time case*. MIT Press, Cambridge, MA, 2004.

[32] H. Van Hasselt A. Guez D. Silver. Deep reinforcement learning with double q-learning. *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[33] G.W. Stewart. On the early history of the singular value decomposition. *SIAM review*, 1993.

[34] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, Massachusetes, 1998.

[35] H.J.C. Berendsen D. van der Spoel R. van Drunen. GROMACS: a message-passing parallel molecular dynamics implementation. *Computer Physics Communications*, 1995.

[36] CJCH Watkins. Learning from Delayed Rewards. *academia edu*, 1989.

[37] J. Ferber G. Weiss. *Multi-agent systems: an introduction to distributed artificial intelligence*. Harlow: Addison Wesley Longman, 1999.

[38] M. Willem. *Minimax theorems*. Springer, 1997.