THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Efficient Solving of POMDPs using Circulant Matrices

AKHIL GUDIPATY
SPRING 2023

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Kenneth Czuprynski
Assistant Research Professor
Applied Research Laboratory
Thesis Supervisor

John Hannan
Associate Professor of Computer Science
Associate Department Head
Associate Professor
Honors Adviser

*Signatures are on file in the Schreyer Honors College.

Abstract

Sequential decision-making problems refer to decision problems that require a series of decisions to be made in a particular order, where the result of each decision can influence the outcome of the following decisions. It is crucial to consider the potential long-term consequences of each decision in these problems. These types of problems are commonly represented as Markov decision processes (MDPs) or partially observable Markov decision processes (POMDPs). This thesis focuses on POMDPS. POMDPs have been used in various areas, including robotics, computer vision, natural language processing, healthcare, finance, logistics, and game AI. In these areas, POMDPs are used to model decision-making problems where there is uncertainty. The primary goal when solving POMDPs is to find an optimal policy; a policy is a representation of the agent's decision making. As part of the search for an optimal policy, a procedure called policy evaluation is necessary. This is typically a computationally, expensive operation. However, if one considered structured policies, the cost can be mitigated. Structured policy representations, such as circulant controllers, have drawn some attention recently as a efficient way to solve POMDPs. In this thesis, we investigate the computational advantages of structured policy representations; specifically, we explore the computational benefits of node major ordering within circulant controller policy representations of POMDPs. We also investigate the use of iterative techniques to expedite the process of solving these equations. The contributions of this thesis are: (1) a derivation of a computationally efficient matrix factorization (2) an iterative scheme which employs the aforemented factorization (3) and a complexity analysis of the derived scheme.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank Dr. Kenneth Czuprynski for supervising my thesis and supporting me throughout this entire process.

I would like to thank Dr. John Hannan for supporting me by being my Thesis Honors advisor.

Chapter 1

Introduction

1.1 Introduction

Sequential making decision problems are decision problems that must be made in a sequence, where the outcome of every decision made can impact the outcome of the subsequent decisions [1]. In these problems, it is important to consider the long-term effects of the decisions. Sequential decision-making problems are often modeled as Markov decision processes (MDPs) or partially observable Markov decision processes (POMDPs), which provide a framework for modeling and solving these types of problems ([2], [3]). Examples of sequential decision-making problems include game playing [4], financial planning [5], medical diagnosis [6], and robot navigation [7].

In a MDP, an agent makes decisions based on its current state. However, in a POMDP, the agent cannot directly observe the environment [8]. We will explore POMDPs further in the next section. Solving POMDPs is difficult because they involve making decisions under uncertainty [9]. In contrast to fully observable MDPs, where the agent has access to the exact state of the environment, POMDPs only provide the agent with a probabilistic observation of the environment state [8]. This leads to a more complex decision-making problem where the agent needs to reason about the uncertainty in the observations to make informed decisions. Furthermore, the data used to solve POMDPs can be extensive, leading to increased computational time to solve the problem [9]. This thesis focuses on a fundamental operation within POMDP solvers called policy evaluation.

Various other approaches have been proposed to expedite the solution process for POMDPs. One such example is reducing the number of belief points required for the solution by employing alternative point-based approximations, which can still uphold the quality of the solution [10]. Another example involves the application of Gaussian Processes to Reinforcement Learning for optimal POMDP dialogue policies, with the aim of accelerating the learning process and obtaining an estimate of the approximation uncertainty [11].

Our work focuses on a particular piece of the optimization problem here called policy evaluation which involves the solution of a large linear system M^{π}. Specfically, we show that when considering a particular type of structured policy representation, M^{π} has exploitable computational structure. The following thesis contributions are made: (1) a computationally efficient matrix factorization is derived, (2) an iterative scheme is proposed that utilizes the derived factorization, and (3) the complexity of the proposed scheme is analyzed.

The remaining portion of this thesis is organized as follows: in the next subsection we formally introduce POMDPs followed by an introduction to circulant matrices and their relevant properties. In Chapter 2 we detail our iterative scheme and Chapter 3 provides a complexity analysis of the derived scheme. We conclude the thesis in chapter 4.

1.2 Background

1.2.1 Introduction to POMDP

Formally, a POMDP can be represented by the tuple $(S, A, T, R, \Omega, O, \gamma)$ [12], where:

- S is a set of states
- A is a set of actions that the agent can take
- T is a set of conditional transition probabilities between states
- Ω is a set of observations
- O is a set of conditional observational probabilities
- $\gamma \in [0, 1)$ is the discount factor
- R is the set of rewards

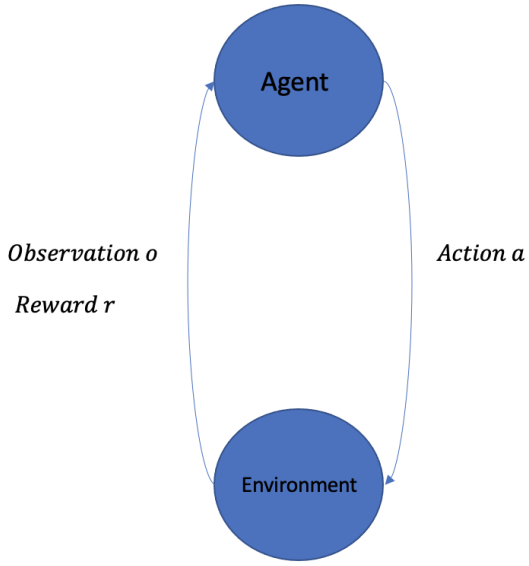

Figure 1.1: POMDP

To understand this better, we can discuss about POMDPs using an example. Consider an agent that is located in a grid-like environment. The agent's task is to navigate to a specific goal location

while avoiding obstacles. The agent can move in four directions $(up, down, left, right)$ at each time step. The agent's observations are limited, and it can only observe a small portion of the environment at a time. This means that the agent does not have complete information about the state of the environment and must make decisions based on its limited observations.

In this example, the agent represents a decision-maker (or agent) navigating through an environment. At each time step, the agent can make an observation about the environment through its sensors and selects an action. Following this the true underlying state is updated probabilistically conditioned on the action taken. The agent's belief about the state of the environment is represented by the probability distribution over the set of states, S.

The agent faces a trade-off between gathering more information about the state of the environment and taking actions that move it closer to the goal. Therefore, the agent has to balance the exploration-exploitation dilemma, where it has to decide whether to explore the environment to gather more information or to exploit the current knowledge to move towards the goal.

To solve this POMDP, the agent uses a policy, represented by π, which maps the current belief state to a probability distribution over the set of actions. There are multiple forms of policy representation. One such representation is a Finite State Controller [13]. An FSC consists of a finite set of nodes, X, a stochastic action selection function, $\psi(x, a)$, and a stochastic successor selection function, $\eta(x, \omega, x')$. The FSC represents the agent's decision-making process and guides it towards the goal.

Associated with a policy is the concept of a value function V^{π}, which evaluates the quality of the policy. Specifically, it gives the expected total reward of a given policy. The value function is used to determine the optimal policy that maximizes the expected total reward. The agent can use various methods, such as value iteration or policy iteration, to compute the optimal policy [14].

In summary, POMDPs model decision-making problems with probabilistic dynamics and partial observability. The agent in this example has to navigate to a specific goal location while avoiding obstacles in a partially observable grid-like environment. The agent uses a policy representation (e.g. FSC) and a value function to make decisions and navigate towards the goal.

At any point in time, the agent is in a state $s \in S$. It takes an action $a \in A$, and the probability of transitioning to a new state s' due to this action is given by $T(s'|s, a)$. The agent also receives an observation o', which depends on the new state s' and the action taken a with probability $O(o'|s', a)$. The set of rewards R then takes in a reward r based on the action performed a from the state s.

In a POMDP, the true state of the system is directly not observed, which is true in a number of applications. This makes POMDPs a powerful modeling tool for applications with uncertainity. POMDPs have been used in a wide range of research areas, including but not limited to:

- Robotics: POMDPs have been used to model decision-making problems in robotics, such as robotic navigation, grasping, and manipulation. For example, a robotic arm in a factory

could use a POMDP to plan its actions based on uncertain sensor readings and uncertain object poses [15].

- Computer vision: POMDPs have been used in computer vision tasks such as object recognition, scene understanding, and tracking. For example, a self-driving car could use a POMDP to make decisions about where to drive based on uncertain sensor readings and uncertain object poses [16].
- Natural Language Processing: POMDPs have been used to model dialogue systems, such as chatbots, which require decision-making based on uncertain user inputs [17].
- Healthcare: POMDPs have been used to model decision-making problems in healthcare, such as treatment planning and drug discovery [18].
- Finance: POMDPs have been used to model decision-making problems in finance, such as portfolio management and risk assessment [19].
- Logistics: POMDPs have been used to model decision-making problems in logistics, such as fleet management and inventory control [20].
- Game AI: POMDPs have been used to model decision-making problems in game AI, such as non-player character behavior and level design [21].

These are just a few examples, POMDPs are widely used in many other fields and research areas as well.

1.2.2 Finite State Controllers

Finite State Controllers (FSCs) are a common way to represent policies for POMDPs due to their efficiency [13]. A FSC policy can be defined as $\pi = \langle X, \psi, \eta \rangle$, where X is the finite set of nodes, $\psi(x, a) = P(a|x)$ is the stochastic action selection function that denotes the probability of selecting action a in node x, and $\eta(x, \omega, x') = P(x'|x, \omega)$ is the stochastic successor selection function that denotes the probability of choosing successor node x' given the observation ω was made after performing node x's action [13].

The value function obtained for a FSC policy representation is given by:

$$V^{\pi}(x, s) = \sum_{a \in A} \psi(x, a)[R(s, a) + \gamma \sum_{s' \in S} \sum_{\omega \in \Omega} O(s', \omega) \sum_{x' \in X} \eta(x, \omega, x') V^{\pi}(x', s')].$$

In order to compute the optimal controller policy for a POMDP, one approach is to use the policy gradient method. In this case, one differentiates the value function with respect to a policy

π and then use the gradient to improve the policy. This process starts by defining a cross-product MDP state transition represented by the matrix $M^\pi \in \mathbb{R}^{|X\times S|\times|X\times S|}$. This matrix describes the transition dynamics of the POMDP, including the probabilities of transitioning between states and observations given the current policy.

To compute the optimal controller policy, we can use a method like gradient ascent [13]. Specifically, we can derive the policy gradient by calculating the gradient of the expected total reward with respect to the policy parameters. The gradient is then used to update the policy parameters in the direction that increases the expected total reward. This process is typically done iteratively until the policy parameters converge to a local maximum. The equation for M^π, the cross-product MDP state transition matrix, is defined as:

$$M^\pi(\langle x, s\rangle, \langle x', s'\rangle) = \sum_{a\in A} \psi(x,a)T(s,a,s') \sum_{w\in\Omega} O(s',\omega)\eta(x,\omega,x').$$

Important to note here that all the matrices and vectors are assumed to be in *node-major* ordering, which we will describe in detail later. Let $r^\pi(\langle x, s\rangle) = \sum_a \psi(x,a)R(s,a)$, we can rewrite the original equation in the following way where v^π is the value function:

$$v^\pi = r^\pi + \gamma M^\pi v^\pi.$$

which can be further written as:

$$v^\pi = (I - \gamma M^\pi)^{-1} r^\pi.$$

Calculating the inverse to solve the system of equations above can be expensive. This paper delves into the topic of structured FSC representation, with a focus on **circulant controllers** [22]. One of the main contributions of this thesis is showing that the matrix M^π happens to have a unique structure that has special properties, which we will discuss in the next section.

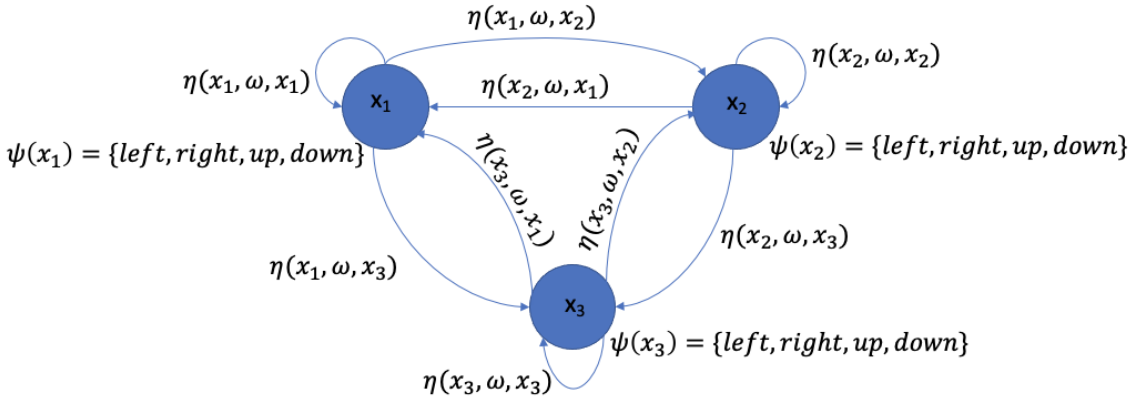

Figure 1.2: Finite State Controller

The diagram above shows the different states that the system can be in and the transitions between them. Each state is represented by a circle, and the transitions between states are represented by arrows. The FSC diagram also includes information about the inputs and outputs of the system, as well as any actions or events that occur when the system transitions between states. The function ψ is the stochastic action selection function that denotes the probability of selecting action a $(left, right, up, down)$ in node x, and η is the stochastic successor selection function that denotes the probability of choosing successor node.

In the next section, we are going to introduce Circulant Matrices and review some relevant properties, which will help us understand the strucutre of M^{π} better.

1.2.3 Introduction to Circulant Matrices

In the previous section, we touched upon the structure of M^{π}. To understand the structure of M^{π}, we must first understand circulant matrices. A circulant matrix is a structured matrix that is constructed by taking a row vector and cyclically shifting its elements to form the rows of the matrix [23]. There are numerous applications for circulant matrices, with the most well-known operation being convolution, which has a range of applications including digital signal processing [24], image compression [25], physics and engineering simulations [26], number theory [27], and cryptography [27].

Given a vector $a = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-1} \end{pmatrix}^T$, a circulant matrix C has the following form:

$$C = \begin{pmatrix} a_0 & a_1 & a_2 & ... & a_{n-1} \\ a_{n-1} & a_0 & a_1 & ... & a_{n-2} \\ . & & & & \\ . & & & & \\ . & & & & \\ a_1 & a_2 & a_3 & ... & a_0 \end{pmatrix}. \tag{1.1}$$

1.2.4 Relationship with Discrete/Fourier Transformation

The Fast Fourier Transform (FFT) is an efficient algorithm for computing the discrete Fourier transform (DFT) of a sequence. The DFT is a powerful tool for analyzing and processing signals or images by expressing them as a sum of complex exponentials of different frequencies [28]. The FFT algorithm significantly reduces the computational cost of DFT and has found wide-spread use in fields such as signal processing, image processing, and scientific computing. Due to its efficiency and versatility, the FFT has become an essential building block in many signal processing algorithms and systems.

Circulant matrices is a subclass of Toeplitz matrices, and they have special properties due to their cyclic structure [23]. Let us consider C from equation (1.1). Specifically, the matrix C has eigenvalue decomposition $C = \frac{1}{n}F^{-1}\Lambda F$, where F is the DFT matrix and Λ is the DFT of the first row of C. Rather than using regular multiplication, multiplying a DFT matrix can be implemented by FFT, which has significant computational advantages. We can use them to solve linear systems fast as well as perform matrix vector multiplication quickly. The $N \times N$ DFT matrix F has the following form:

$$F = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_N^1 & \omega_N^2 & \dots & \omega_N^{N-1} \\ 1 & \omega_N^2 & \omega_N^4 & \dots & \omega_N^{2(N-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_N^{N-1} & \omega_N^{2(N-1)} & \dots & \omega_N^{(N-1)^2} \end{pmatrix}. \tag{1.2}$$

1.2.5 Solving a system of equations for circulant matrices

To illustrate the computational property of circulant matrices, we consider the solution for the linear system $Cx = b$, where C is an $n \times n$ Circulant matrix, $x \in \mathbb{R}^n$ and $b \in \mathbb{R}^n$. Now, the main idea here is to solve for the unknown variable x in $Cx = b$. We are going to discuss two methods to solve this:

1. The naive method using matrix inversion

2. The efficient method using FFTs

The naive method

The naive way is quite simple. We can find x by simply computing the inverse of the matrix C, and then multiplying it with b

$$x = C^{-1}b.$$

The cost of computing the inverse of a matrix, also known as matrix inversion, should be considered. Matrix inversion is typically performed using Gaussian elimination, which can be completed in $O(n^3)$. This can become prohibitive for large matrices.

The efficient method using FFTs [22]

Utilizing our understanding of the connection between discrete Fourier transforms (DFTs) and circulant matrices, we can solve the system by following a series of steps. The first step in this method is to generate the DFT matrix F_n, which should have the same dimensions as C. While the inverse of the DFT matrix can be calculated, it is computationally expensive as mentioned in a previous section. Therefore, we can avoid matrix inversion by calculating the inverse by taking the conjugate transpose of the DFT matrix F_n and multiplying it by $\frac{1}{n}$. This is possible because an appropriately scaled version of F_n belongs to a class of matrices known as unitary matrices, which have the property that their inverse is equal to their conjugate transpose.

The next step is to find Λ, which is equal to $diag(F_n c_r)$. c_r is the first row of C. Now, using $C = \frac{1}{n}({F_n}^{-1}\Lambda F_n)$, we can rewrite our original equation by substituting for C:

$$(\frac{1}{n}{F_n}^{-1}\Lambda F_n)x = b.$$

After converting our original equation into the following equation, we can further break down the solution for this equation into three steps.

Step 1: Introduce a new variable $\tilde{b}$ where $\tilde{b}$ is equal to $nF_n b$. Compute $\tilde{b}$.

Step 2: Compute $F_n x = (\Lambda^{-1})\tilde{b}$.

Step 3: Compute $x = F_n^{-1}(\Lambda^{-1}\tilde{b})$.

The above method allows us to obtain x in a more efficient manner compared to our first method. This is due to the fact that we do not need to calculate the inverse using Gaussian elimination. Every multiplication by the matrix F_n and F_n^{-1} can be thought of as taking the fast Fourier

transform of whatever vector your are multiplying by. This illustrates how the properties of Circulant matrices can be leveraged in matrix computations. In considering circulant controllers, this thesis will show that some of the computational properites can be leveraged to perform efficient policy evaluation within POMDP algorithm. In the next chapter, we are going to be exploring circulant controllers, which was introduced in the previous section.

Chapter 2

Iterative scheme for circulant controllers

2.1 Circulant Controllers

After discussing circulant matrices in the previous chapter, we can now explore their relevance to POMDPs and how the structure of M^π relates to them, given our understanding of the properties of circulant matrices.

A circulant controller is a type of finite state controller in control theory that has a circulant structure, meaning the coefficients in the matrix representation of the controller are cyclic shifts of one another [22]. Circulant controllers can be used in control systems that have a repetitive structure or are periodic in nature, and they are particularly useful when it is desired to simplify the design of the controller, reduce the computational complexity of the control algorithm, or achieve a desired frequency response in the control loop [22].

This thesis will show that the circulant structure of the controller impacts the underlying structure of the cross product MDP matrix. This can result in significant computational savings, as well as ease of design, compared to other types of finite state controllers. Additionally, circulant controllers can provide robust performance, since the cyclic structure of the controller ensures that its behavior is consistent over time, even in the presence of perturbations or noise in the system [22].

Referring back to Chapter 1, we saw that the equation for M^π, the cross-product MDP state transition matrix, is defined as:

$$M^\pi(\langle x, s\rangle, \langle x', s'\rangle) = \sum_{a\in A} \psi(x,a)T(s,a,s') \sum_{w\in\Omega} O(s',\omega)\eta(x,\omega,x'),$$

where x is a controller node and s is a state in the underlying POMDP. We also mentioned that all matrices and vectors are chosen to be in *node-major* ordering. The node-major ordering in the context of the M^π matrix refers to the way the matrix is indexed by the ordered pairs $(\langle x, s\rangle, \langle x', s'\rangle)$; a choice for the logical ordering is necessary. Node major ordering is a way of grouping the node indexing together. This ordering has yet to be considered in the context of circulant controllers and is the focus of this thesis.

By using node major ordering, for each fixed state s_j we enumerate all the controller nodes. This is akin to indexing a 2D array with a double for loop where the outer loop corresponds to the states and the inner loop the nodes:

```
for j=1 to num-state:
    for i=1 to num-nodes:
        <x_i, s_j>
    end
end
```

So for example, if $|X| = 3$ and $|S| = 2$, we map the ordered pairs $\langle x_i, s_j \rangle$ to a vector as follows:

$$2D - representation : \begin{pmatrix} \langle x_1, s_1 \rangle & \langle x_1, s_2 \rangle \\ \langle x_2, s_1 \rangle & \langle x_2, s_2 \rangle \\ \langle x_3, s_1 \rangle & \langle x_3, s_2 \rangle \end{pmatrix},$$

$$1D - representation : \begin{pmatrix} \langle x_1, s_1 \rangle \\ \langle x_2, s_1 \rangle \\ \langle x_3, s_1 \rangle \\ \langle x_1, s_2 \rangle \\ \langle x_2, s_2 \rangle \\ \langle x_3, s_2 \rangle \end{pmatrix}.$$

By using node-major ordering, the matrix M^π will have specific structure when considering circulant controllers. This thesis investigates that structure and the computational benefits that it has. Recall that the value function obtained for a FSC policy representation is given by:

$$v^\pi = (I - \gamma M^\pi)^{-1} r^\pi$$

where r^π is the reward function, $0 < \gamma < 1$ the discount factor, and v^π the value function. We know that calculating the inverse of a matrix can be a computationally intensive task, particularly for large matrices. The size and complexity of the matrix, as well as the method used to calculate its inverse, can all contribute to the computational resources required. This is why it is important to consider the computational costs when determining the most appropriate method for calculating the inverse of a matrix, especially in real-time control systems where computational efficiency is crucial.

M^π has a unique structure that will allow us to factorize it into two separate matrices. We can use this factorization to develop a fast iterative scheme which fits within the framework of a canonical Jacobi splitting. This matrix splitting will allow us to solve this system iteratively. We begin by establishing some of the relevant properties of circulant matrices needed by this. First, the sum of two circulant matrices is a circulant matrix [29]. For completeness, we state the lemma and prove it here.

Lemma: Let A and B be circulant matrices. Then $A + B$ is a circulant matrix.

Proof: Let us assume we have two circulant matrices A and B of size $n \times n$. A circulant matrix has the property that each row is a cyclic shift of the row above it. When we add two circulant matrices, each element in the resulting matrix is simply the sum of the corresponding elements in

the original matrices. So given

$$A = \begin{pmatrix} a_0 & a_1 & a_2 & \dots & a_{n-2} & a_{n-1} \\ a_{n-1} & a_0 & a_1 & \dots & a_{n-3} & a_{n-2} \\ a_{n-2} & a_{n-1} & a_0 & \dots & a_{n-4} & a_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 & a_3 & a_4 & \dots & a_0 & a_1 \\ a_1 & a_2 & a_3 & \dots & a_{n-1} & a_0 \end{pmatrix} \quad B = \begin{pmatrix} b_0 & b_1 & b_2 & \dots & b_{n-2} & b_{n-1} \\ b_{n-1} & b_0 & b_1 & \dots & b_{n-3} & b_{n-2} \\ b_{n-2} & b_{n-1} & b_0 & \dots & b_{n-4} & b_{n-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ b_2 & b_3 & b_4 & \dots & b_0 & b_1 \\ b_1 & b_2 & b_3 & \dots & b_{n-1} & b_0 \end{pmatrix}$$

the addition of A and B yields:

$$A + B = \begin{pmatrix} a_0 + b_0 & a_1 + b_1 & a_2 + b_2 & \dots & a_{n-2} + b_{n-2} & a_{n-1} + b_{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_2 + b_2 & a_3 + b_3 & a_4 + b_4 & \dots & a_0 + b_0 & a_1 + b_1 \\ a_1 + b_1 & a_2 + b_2 & a_3 + b_3 & \dots & a_{n-1} + b_{n-1} & a_0 + b_0 \end{pmatrix}.$$

Since each row in the sum of two circulant matrices is simply a shifted version of the row above it, the sum of two circulant matrices is also a circulant matrix. In other words, the property of being a circulant matrix is preserved when adding two circulant matrices. □

Now that we established circulant matrices are closed under addition, we will look at how the cross product MDP matrix M^π can be factored into the product of a block matrix of diagonal blocks and a block diagonal matrix of circulant blocks. This is one of the primary contributions of this thesis and is what allows us to derive an efficient iterative scheme within the canonical Jacobi splitting.

Theorem: Given a circulant controller, the MDP state transition matrix has the factorization

$$M^\pi = DC$$

where D is a block matrix of diagonal blocks and C is block diagonal matrix of circulant blocks. Proof: First we express M^π as the block matrix of form:

$$M^\pi = \begin{pmatrix} M_{s_1 s_1} & M_{s_1 s_2} & M_{s_1 s_1} & \dots & M_{s_1 s_{|S|}} \\ M_{s_2 s_1} & M_{s_2 s_2} & M_{s_2 s_3} & \dots & M_{s_2 s_{|S|}} \\ \vdots & & & & \vdots \\ M_{s_{|S|} s_1} & M_{s_{|S|} s_1} & M_{s_{|S|} s_3} & \dots & M_{s_{|S|} s_{|S|}} \end{pmatrix}.$$

Each block in the matrix M^π denotes a $|X| \times |X|$ matrix. Note that M^π has this specific blocking

due to our choice of node-major ordering. Each $M_{s_i s_j}(x_k, x_l)$ is in the form:

$$M_{s_i s_j}(x_k, x_l) = \sum_{a \in A} \psi(a, x_k) T(s, a, s') \sum_{w \in \Omega} O(s', \omega) \eta(x_k, w, x_l).$$

Now, let us define two different matrices. The block diagonal matrix $D_{s,s'}$, of size $|X| \times |X|$, can be defined in the following way:

$$D_{s,s'}(x_i) = \sum_{a \in A} \psi(a, x_i) T(s, a, s'),$$

where x_i is a controller node. The matrix C, of size $|X| \times |X|$, can be defined in the following way:

$$C_{s,s'}(x_i, x_j) = \begin{cases} \sum_{w \in \Omega} O(s', \omega) \eta(x_i, w, x_j) & \text{if } s = s' \\ 0 & \text{if } s \neq s' \end{cases}. \tag{2.1}$$

Each $C_{s_i s_i}$ is a circulant matrix. We know that C is dependent on Ω and η from equation 2.1. We also know that for a fixed ω, η is a circulant matrix. From equation 2.1

$$C_{s,s'}(x_i, x_j) = \sum_{w \in \Omega} O(s', \omega) \eta(x_i, w, x_j).$$

Provided that $s = s'$. Since s' is fixed, we can view the coefficient of a β as an observation dependent constant β_ω. Then $C_{s,s'} = \sum_{w \in \Omega} \beta \eta(., w, .)$. Since $\eta(., w, .)$ is a circulant, we have that $C_{s,s'}$ is the sum of circulant matrices and thus from the lemma, we know that sum of circulant matrices is a circulant matrix. Therefore, $C_{s,s'}$ is a circulant matrix.

Consider the product:

$$\begin{pmatrix} D_{s_1 s_1} & D_{s_1 s_2} & D_{s_1 s_3} & ... & D_{s_1 s_{|S|}} \\ D_{s_2 s_1} & D_{s_2 s_2} & D_{s_2 s_3} & ... & D_{s_2 s_{|S|}} \\ \vdots & & & & \vdots \\ D_{s_{|S|} s_1} & D_{s_{|S|} s_2} & D_{s_{|S|} s_3} & ... & D_{s_{|S|} s_{|S|}} \end{pmatrix} \begin{pmatrix} C_{s_1 s_1} & 0 & 0 & ... & 0 \\ 0 & C_{s_2 s_2} & 0 & ... & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & ... & C_{s_{|S|} s_{|S|}} \end{pmatrix},$$

where each $D_{s_i s_j}$ is a diagonal matrix of dimension $|X| \times |X|$ and each $C_{s_i s_i}$ is a circulant matrix of dimension $|X| \times |X|$. Multiplying these out, we have:

$$\begin{pmatrix} D_{s_1 s_1} C_{s_1 s_1} & D_{s_1 s_2} C_{s_2 s_2} & D_{s_1 s_3} C_{s_3 s_3} & ... & D_{s_1 s_{|S|}} C_{s_{|S|} s_{|S|}} \\ D_{s_2 s_1} C_{s_1 s_1} & D_{s_2 s_2} C_{s_2 s_2} & D_{s_2 s_3} C_{s_3 s_3} & ... & D_{s_2 s_{|S|}} C_{s_{|S|} s_{|S|}} \\ \vdots & & & & \vdots \\ D_{s_{|S|} s_1} C_{s_1 s_1} & D_{s_{|S|} s_2} C_{s_2 s_2} & D_{s_{|S|} s_3} C_{s_3 s_3} & ... & D_{s_{|S|} s_{|S|}} C_{s_{|S|} s_{|S|}} \end{pmatrix}$$

We now show that $M_{s_i s_j} = D_{s_i s_j} C_{s_i s_j}$. We can see that every element in this matrix now is a matrix product, and let us take the arbitrary element to see what it looks like. Let us look at $D_{s_i s_j} C_{s_j s_j}$.

$$D_{s_i s_j} C_{s_j s_j} = \begin{pmatrix} D_{s_i s_j}(x_1) & 0 & 0 & ... & 0 \\ 0 & D_{s_i s_j}(x_2) & 0 & ... & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & ... & D_{s_i s_j}(x_{|X|}) \end{pmatrix} \begin{pmatrix} C_{s_j s_j}(x_1, x_1) & C_{s_j s_j}(x_1, x_2) & ... & C_{s_j s_j}(x_1, x_{|X|}) \\ C_{s_j s_j}(x_2, x_1) & C_{s_j s_j}(x_2, x_2) & ... & C_{s_j s_j}(x_2, x_{|X|}) \\ \vdots & & & \vdots \\ C_{s_j s_j}(x_{|X|}, x_1) & C_{s_j s_j}(x_{|X|}, x_2) & ... & C_{s_j s_j}(x_{|X|}, x_{|X|}) \end{pmatrix}.$$

Multiplying this will give us:

$$D_{s_i s_j} C_{s_j s_j} = \begin{pmatrix} D_{s_i s_j}(x_1) C_{s_j s_j}(x_1, x_1) & D_{s_i s_j}(x_1) C_{s_j s_j}(x_1, x_2) & ... & D_{s_i s_j}(x_1) C_i(x_1, x_{|x|}) \\ D_{s_i s_j}(x_2) C_{s_j s_j}(x_2, x_1) & D_{s_i s_j}(x_2) C_{s_j s_j}(x_2, x_2) & ... & D_{s_i s_j}(x_2) C_{s_j s_j}(x_2, x_{|x|}) \\ \vdots & & & \vdots \\ D_{s_i s_j}(x_{|x|}) C_{s_j s_j}(x_{|x|}, x_1) & D_{s_i s_j}(x_{|x|}) C_{s_j s_j}(x_{|x|}, x_2) & ... & D_{s_i s_j}(x_{|x|}) C_{s_j s_j}(x_{|x|}, x_{|x|}) \end{pmatrix}.$$

If we take one element and expand this we can see that it is in the form:

$$D_{s_i s_j}(x_k) C_{s_j s_j}(x_k, x_l) = \sum_{a \in A} \psi(a, x_k) T(s, a, s') \sum_{w \in \Omega} O(s', \omega) \eta(x_k, \omega, x_l).$$

From this we can conclude that $M_{s_i s_j} = D_{s_i s_j} C_{s_i s_j}$ from which it follows that

$$M^\pi \begin{pmatrix} D_{s_1 s_1} & D_{s_1 s_2} & D_{s_1 s_3} & ... & D_{s_1 s_{|S|}} \\ D_{s_2 s_1} & D_{s_2 s_2} & D_{s_2 s_3} & ... & D_{s_2 s_{|S|}} \\ \vdots & & & & \vdots \\ D_{s_{|S|} s_1} & D_{s_{|S|} s_2} & D_{s_{|S|} s_3} & ... & D_{s_{|S|} s_{|S|}} \end{pmatrix} \begin{pmatrix} C_{s_1 s_1} & 0 & 0 & ... & 0 \\ 0 & C_{s_2 s_2} & 0 & ... & 0 \\ \vdots & & & & \vdots \\ 0 & 0 & 0 & ... & C_{s_{|S|} s_{|S|}} \end{pmatrix}$$

We have now shown that given a circulant controller and a node major ordering, the matrix M^π has a specific factorization. We would like to exploit this structure in our iterative scheme. □

The next lemma will establish we have a similar factorization when considering a splitting of the matrix M^π.

Lemma: Consider the splitting of $M^\pi = M_D^\pi + L + U$, where M_D^π is a diagonal matrix, L is a lower triangualr matrix, and U is an upper triangular matrix, then $L + U = DC$, where D is a block matrix of diagonal blocks and C is a block diagonal matrix with circulant blocks.

Proof: We can rewrite M^π as:

$$M^\pi = M_D^\pi + L + U.$$

Where M_D^π denotes the blocks along the diagonal in M^π. We can rewrite our equation as:

$$L + U = M^{\pi} - M_D^{\pi}.$$

From our previous proof we showed that we can write M^{π} as DC. Let us consider a sub block in $M^{\pi} - M_D^{\pi}$.

$$M_{ij}^{\pi} - M_{Dij}^{\pi} = \begin{pmatrix} d_1c_1 & d_1c_2 & d_1c_3 & ... & d_1c_n \\ d_2c_n & d_2c_1 & d_2c_2 & ... & d_2c_{n-1} \\ \vdots & & & & \\ d_nc_2 & d_nc_3 & d_nc_4 & ... & d_nc_1 \end{pmatrix} - \begin{pmatrix} d_1c_1 & 0 & 0 & ... & 0 \\ 0 & d_2c_1 & 0 & ... & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & ... & d_nc_1 \end{pmatrix},$$

$$M_{ij}^{\pi} - M_{Dij}^{\pi} = \begin{pmatrix} 0 & d_1c_2 & d_1c_3 & ... & d_1c_n \\ d_2c_n & 0 & d_2c_2 & ... & d_2c_{n-1} \\ \vdots & & & & \\ d_nc_2 & d_nc_3 & d_nc_4 & ... & 0 \end{pmatrix}.$$

This will result in:

$$M_{ij}^{\pi} - M_{Dij}^{\pi} = \begin{pmatrix} d_1 & 0 & 0 & ... & 0 \\ 0 & d_2 & 0 & ... & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & ... & d_n \end{pmatrix} \begin{pmatrix} 0 & c_2 & c_3 & ... & c_n \\ c_n & 0 & c_2 & ... & c_{n-1} \\ \vdots & & & & \\ c_2 & c_3 & c_4 & ... & 0 \end{pmatrix}.$$

Where $D_{ij} = \begin{pmatrix} d_1 & 0 & 0 & ... & 0 \\ 0 & d_2 & 0 & ... & 0 \\ \vdots & & & & \\ 0 & 0 & 0 & ... & d_n \end{pmatrix}$ is a diagonal matrix, and $C_{ij} = \begin{pmatrix} 0 & c_2 & c_3 & ... & c_n \\ c_n & 0 & c_2 & ... & c_{n-1} \\ \vdots & & & & \\ c_2 & c_3 & c_4 & ... & 0 \end{pmatrix}$ is a circulant matrix. Hence, proved. □

2.2 An efficient iterative scheme

In this section, we will present an efficient method for solving systems of equations involving circulant block matrices using Jacobi iteration. Jacobi method is a very popular iterative method that we are going to make use in our solution. Before delving into the specifics of the method, it is helpful to provide an overview of Jacobi iteration and its role in estimating solutions to systems of equations.

2.2.1 Jacobi Method

The Jacobi method is a type of fixed point iteration that can be used to approximate solutions to systems of equations [30]. To implement this method, the first step is to rewrite the equations in a way that allows us to solve for the unknowns. Specifically, we solve the ith equation for the ith unknown. This process is then repeated multiple times in order to improve the accuracy of the solution [30].

The above process can also be concisely represented in matrix form. The basic idea behind the Jacobi method is to decompose the matrix A into a diagonal matrix S, a strictly lower triangular matrix L, and a strictly upper triangular matrix U, such that $A = S + L + U$. The decomposition can be represented as follows:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} a_{11} & 0 & 0 & \dots & 0 \\ 0 & a_{22} & 0 & \dots & 0 \\ \vdots & & & \ddots & \\ 0 & 0 & 0 & \dots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ a_{21} & 0 & 0 & \dots & 0 \\ \vdots & & & & \\ a_{n1} & a_{n2} & a_{n3} & \dots & 0 \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & 0 & a_{23} & \dots & a_{2n} \\ \vdots & & & & \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Once the matrix A has been decomposed in this way, the Jacobi method proceeds by iteratively solving for the x values in $Ax = b$ in the following way:

$$\begin{aligned} Ax &= b \\ (S + L + U)x &= b \\ Sx &= b - (L + U)x \\ x &= S^{-1}(b - (L + U)x). \end{aligned}$$

This gives rise to the iteration

$$x^{k+1} = S^{-1}(b - (L + U)x^k).$$

Which is the Jacobi method [30].

2.2.2 Efficient Policy Evaluation

Now, we will show how Jacobi iteration can be combined with our matrix factorization to derive our efficient scheme for policy evaluation with circulant controllers. We derive our iterative scheme by using Jacobi iteration and show that we maintain some structure of the MDP state transition matrix M^π discussed in Chapter 1.

Our goal is to solve this value function obtained for a FSC policy representation is given by:

$$(I - \gamma M^\pi)v^\pi = r^\pi. \tag{2.2}$$

Instead of applying the splitting to $(I - \gamma M^\pi)$, we first write

$$M^\pi = M_D^\pi + L + U,$$

where M_D^π are the diagonal entries of M^π. Employing our previously established lemma we have $L + U = DC$, where D is a block diagonal matrix and C is a diagonal block matrix with circulant blocks. We then have

$$M^\pi = M_D^\pi + DC$$

from our previous lemma. Substituting into (2.2), we have:

$$(I - \gamma(M_D^\pi + DC))v^\pi = r^\pi.$$

Which can be reorganized as:

$$(I - \gamma M_D^\pi)v^\pi = r^\pi + \gamma DCv^\pi.$$

This gives rise to the iteration:

$$v_{k+1}^\pi = (I - \gamma M_D^\pi)^{-1}(r^\pi + \gamma DCv_k^\pi). \tag{2.3}$$

Here, M_D^π represents the diagonal of the matrix M^π and DC is a diagonal matrix.

In each iteration, we update each element of the unknown vector v_{k+1}^π based on the corresponding elements of the current approximation v_k^π and the right-hand side of the equation.

We iterate (2.3) until the approximation v_{k+1}^π converges to the solution, or until a maximum number of iterations has been reached. We also saw that in Chapter 1, that circulant matrices have exceptional computational properties. This is the whole point of the DC factorization and the reason why this method is very efficient. Our scheme in (2.3) provides the solution of (2.2); i.e. it performs policy evaluation through the defined iteration. The main goal of the thesis has

been achieved through this iterative scheme. However, it remains to be demonstrated that this approach is faster compared to traditional methods used for policy evaluation. Let us look at the time complexity and see how it is faster than the traditional method.

Chapter 3

Analyzing time complexity

3.1 Introduction

In this chapter, we investigate the time complexity of the methods discussed into the previous chapters and then conclude as to why they are faster than the traditional methods.

Recall policy evaluation is looking to solve (2.2):

$$(I - \gamma M^{\pi})v^{\pi} = r^{\pi}.$$

3.2 Gaussian Elimination

Based on our analysis in Chapter 1, it is clear that the dominant process in this method is Gaussian elimination, which has a time complexity of $O(n^3)$ when n is the size of the matrix [31]. This complexity applies to both regular circulant matrices and block circulant matrices if their underlying structure isn't exploited. As a result, computing the value function by solving

$$(I - \gamma M^{\pi})v^{\pi} = r^{\pi}$$

via Gaussian elimination is cubic in the dimensions of M^{π}. Since M^{π} is $|X||S| \times |X||S|$, evaluating a policy via Gaussian Elimination is $O(|X|^3|S|^3)$. This is one of the primary methods used for policy evaluation if the structure of M^{π} is not available. This provides a baseline to compare against for the performance of our method. However, standard jacobi method or any iterative solver can further reduce this to $O(n^2)$, which is another baseline for us [31].

3.3 Our iterative scheme

In this section, we analyze the time complexity of Jacobi iteration when we take into account the matrix structure of M^{π}:

$$M^{\pi} = M_D^{\pi} + DC.$$

By applying our method to equation (2.2), we obtained the iteration:

$$v_{k+1}^{\pi} = (I - \gamma M_D^{\pi})^{-1}(r^{\pi} + \gamma DC v_k^{\pi}). \tag{3.1}$$

Additionally, we discussed the reason for splitting was that computing the inverse of a matrix can be very expensive. Eliminating the step of computing the inverse of a matrix can save us a lot of time. If we take a closer look at this method, we haven't completely eliminated the step of inverting a matrix. However, we have simplified it further. We only need to compute the inverse of $(I - \gamma M_D^{\pi})^{-1}$, which is a diagonal matrix. Let us divide the entire method into separate processes

1. Computing the inverse of $(I - \gamma M_D^\pi)^{-1}$.

2. Computing the matrix multiplication DCv_k^π.

These are the most dominating costs in computing a single iteration of equation 3.1. Note that we are only computing the time complexity for one iteration but it is assumed that the number of iterations required is much smaller than the dimension of the system. In considering the inverse of $I - \gamma M_D^\pi$, we recall that both are diagonal matrices. As a result, the inverse is linear in the dimension. In this case, $O(|X||S|)$. Next, we consider DCv_k^π. This term is more complicated to analyze. We need to consider the block structure of D and C. We have

$$DC = \begin{pmatrix} D_{s_1s_1}C_{s_1s_1} & D_{s_1s_2}C_{s_2s_2} & ... & D_{s_1s_{|S|}}C_{s_{|S|}s_{|S|}} \\ \vdots & & ... & \\ D_{s_{|S|}s_1}C_{s_1s_1} & & ... & D_{s_{|S|}s_S}C_{s_{|S|}s_{|S|}} \end{pmatrix},$$

which has total dimension $(|X||S|) \times (|X||S|)$ and each block is $D_{s_is_j}C_{s_is_j}$. For a given vector $\vec{x}$ $\in \mathbb{R}^{|X||S|}$, multiplication of DC by $\vec{x}$ has the form:

$$DC\vec{x} = \begin{pmatrix} D_{s_1s_1}C_{s_1s_1}x_1 + D_{s_1s_2}C_{s_2s_2}x_2 + ... + D_{s_1s_{|S|}}C_{s_{|S|}s_{|S|}}x_{|S|} \\ D_{s_2s_1}C_{s_1s_1}x_1 + D_{s_2s_2}C_{s_2s_2}x_2 + ... + D_{s_2s_{|S|}}C_{s_{|S|}s_{|S|}}x_{|S|} \\ \vdots \\ D_{s_{|S|}s_1}C_{s_1s_1}x_1 + D_{s_{|S|}s_2}C_{s_2s_2}x_2 + ... + D_{s_{|S|}s_{|S|}}C_{s_{|S|}s_{|S|}}x_{|S|} \end{pmatrix},$$

where our vector $\vec{x} = \left(x_1, x_2, \ldots, x_{|S|}\right)^T \in \mathbb{R}^{|X||S|}$ and each $x_k \in \mathbb{R}^{|X|}$. As a result if we analyze the complexity of the individual block multiplications we will be able to obtain the full complexity of computing $DC\vec{x}$.

In each case $D_{s_is_j}$ is a diagonal matrix of dimensions $|X| \times |X|$ and $C_{s_js_j}$ is a circulant matrix of dimensions $|X| \times |X|$. For simplicity of notation, we can omit the subscripts and analyze DCx. Using the eigenvalues decomposition of a circulant matrix, we have

$$DCx = D\frac{1}{n}F_n^{-1}\Lambda F_n x.$$

Multiplication of F_n by x can be done via the FFT which is $O(|X|log|X|)$ given the dimensions of F_n. Letting $y = F_n x$, we have

$$DCx = D\frac{1}{n}F_n^{-1}\Lambda y.$$

The matrix Λ is diagonal so multiplication of Λy is $O(|X|)$. Letting $z = \Lambda y$, we have

$$DCx = D\frac{1}{n}F_n^{-1}z.$$

Note that $n = |X|$. Multiplication of $F_n^{-1}z$ can again be done via FFT, which is $O(|X|log|X|)$. The final matrix is again diagonal which is $O(|X|)$. As a result the computation of a single block multiplication is $O(|X|log|X|+|X|)$ which is improvement over $O(|X|^2)$ which is the complexity if the traditional Jacobi mwthod is applied, which is another baseline of comparison for us.

Taking the result in context of 3.1 we have that each of the block multiplications $D_{s_is_j}C_{s_js_j}x_j$ is $O(|X|log|X| + |X|)$. In total we perform $|S|^2$ of these operations. As a result, the multiplications in our Jacobi iteration DCv^π is $O(|S|^2|X|log|X| + |S|^2|X|)$. Again, this is an improvement as the traditional matrix multiplication of an $|S||X| \times |S||X|$ matrix which is $O(|S|^2|X|^2)$. Assuming the number of iterations required by Jacobi Iteration is small compared to the size of M^π, this scheme significantly outperforms.

We can look at this time complexity visually to see that our proposed method is better than the unmodified Jacobi method.

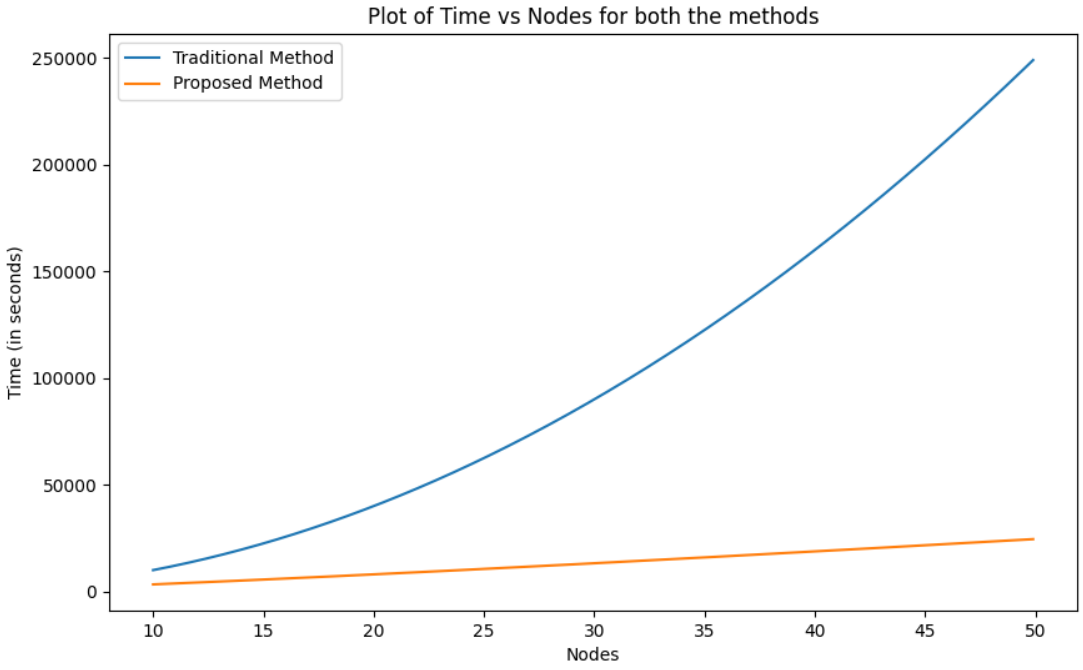

Figure 3.1: Time vs Nodes when the number of states = 10

By looking at the plot, we can understand the relationship between the number of states, number of nodes, and the time for one iteration.

Chapter 4

Conclusion

In this thesis we considered ways to speed up POMDP solvers by using structured policy representations. We showed that a matrix factorization involving circulant matrices was possible when considering circulant FSC policy representation. Based on this factorization, we developed an iterative scheme that leverages this structure to achieve computational efficiency. Our proposed scheme was shown to have superior computational complexity when compared with a naive implementation of Jacobi's method, which is commonly used for solving linear systems. Overall, our work shows that exploiting the structure of policy representations can lead to significant improvements in the computational efficiency of POMDP solvers. This has important implications for a range of applications in which POMDPs are used, including robotics, autonomous systems, and decision-making under uncertainty. This thesis has explored efficient matrix computations in the context of circulant controllers. In the furture, one can consider other structured policy representations and their impact of the matrix structure of policy evaluation.

Bibliography

[1] Andrew Gehret Barto, Richard S Sutton, and C. J. C. H. Watkins. *Learning and sequential decision making*. University of Massachusetts, 1989.

[2] Mykel J. Kochenderfer, Tim A. Wheeler, and Kyle H. Wray. *Algorithms for Decision Making*. MIT Press, 2022.

[3] Régis Sabbadin. A possibilistic model for qualitative sequential decision problems under uncertainty in partially observable environments. *arXiv preprint arXiv:1301.6736*, 2013.

[4] David Silver and Joel Veness. Monte-carlo planning in large pomdps. In *Advances in neural information processing systems*, volume 23, 2010.

[5] Benjamin Ng, Carl Meyers, Kofi Boakye, and Jeffrey Nitao. Towards applying interactive pomdps to real-world adversary modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pages 1814–1820, July 2010.

[6] Luchen Li, Matthieu Komorowski, and Aldo A. Faisal. The actor search tree critic (astc) for off-policy pomdp learning in medical decision making. *arXiv preprint arXiv:1805.11548*, 2018.

[7] Amalia Foka and Panos Trahanias. Real-time hierarchical pomdps for autonomous robot navigation. *Robotics and Autonomous Systems*, 55(7):561–571, 2007.

[8] Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving markov decision problems. *arXiv preprint arXiv:1302.4971*, 2013.

[9] Nicolas Meuleau, David Hsu, Peter Kim, Thore Chen, and Valentina Lu. Solving pomdps by searching the space of finite policies. *arXiv preprint arXiv:1301.6720*, 2013.

[10] Finale Doshi and Nicholas Roy. The permutable pomdp: Fast solutions to pomdps for preference elicitation, 2008. Retrieved March 26, 2023, from https://www.ifaamas.org/Proceedings/aamas08/proceedings/pdf/paper/AAMAS08$_0$244.pdf.

[11] M. Gasic, F. Jurcicek, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young. Gaussian processes for fast policy optimisation of pomdp-based dialogue managers. In *Proceedings of the SIGDIAL 2010 Conference*, pages 201–204, 2010.

[12] G. Hollinger. Partially observable markov decision processes (pomdps). https://www.cs.cmu.edu/ ggordon/780-fall07/lectures/POMDP$_lecture.pdf, 2007. Accessed on March 30, 2023.$

[13] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Advances in neural information processing systems*, volume 16, 2003.

[14] Daniel S Bernstein, Eric A Hansen, and Shlomo Zilberstein. Bounded policy iteration for decentralized pomdps. In *Proceedings of the nineteenth international joint conference on artificial intelligence (IJCAI)*, pages 52–57, 2005.

[15] Brian C. Williams and Wendy A. Rogers. A survey of point-based pomdp solvers. 2003.

[16] J. H. Rhee, W. J. Sung, M. Y. Nam, H. Byun, and P. K. Rhee. An efficient eye tracking using pomdp for robust human computer interaction. In L. Nalpantidis, V. Krüger, J. O. Eklundh, and A. Gasteratos, editors, *Computer Vision Systems: ICVS 2015*, pages 477–486. Springer, Cham, 2015.

[17] X. Xiang and S. Foo. Recent advances in deep reinforcement learning applications for solving partially observable markov decision processes (pomdp) problems: Part 1—fundamentals and applications in games, robotics and natural language processing. *Machine Learning and Knowledge Extraction*, 3(3):554–581, 2021.

[18] Amulya Yadav, Leandro Marcolino, Eric Rice, Robin Petering, Hailey Winetrobe, Harmony Rhoades, Milind Tambe, and Heather Carmichael. Psinet - an online pomdp solver for hiv prevention in homeless populations. In *In AAAI-15 Workshop on Planning, Search, and Optimization (PlanSOpt-15)*, 2015.

[19] Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

[20] John Smith. A pomdp framework for coordinated guidance of autonomous uavs for multitarget tracking. *Journal of Aerospace Engineering*, 16(3):132–140, May 2008.

[21] C. T. Tan and H.-lun Cheng. Implant: An integrated MDP and POMDP learning agent for adaptive games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 5, pages 94–99, 2009.

[22] Kyle H. Wray and Kenneth Czuprynski. Scalable pomdp decision-making using circulant controllers. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6831–6837, Xi'an, China, 2021.

[23] Robert M Gray. Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239, 2006.

[24] Wenling Zhao. Notice of retraction: The inverse problem of anti-circulant matrices in signal processing. In *2009 Pacific-Asia Conference on Knowledge Engineering and Software Engineering*, pages 47–50, 2009.

[25] Eunice Carrasquinha, Conceicao Amado, Ana M Pires, and Lina Oliveira. Image reconstruction based on circulant matrices. *Signal Processing: Image Communication*, 63:72–80, 2018.

[26] Jingchao Zhang, Ning Fu, and Xiyuan Peng. Compressive circulant matrix based analog to information conversion. *IEEE Signal Processing Letters*, 21(4):428–431, 2014.

[27] Philip J Davis. *Circulant matrices*, volume 120. Wiley New York, 1979.

[28] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[29] Nick Higham. What is a circulant matrix? https://nhigham.com/2022/09/27/what-is-a-circulant-matrix/: :text=FurthermoreOctober 13 2022. Accessed on March 30, 2023.

[30] Davod Khojasteh Salkuyeh. Generalized jacobi and gauss-seidel methods for solving linear system of equations. *NUMERICAL MATHEMATICS-ENGLISH SERIES*-, 16(2):164, 2007.

[31] Lloyd N Trefethen. Three mysteries of gaussian elimination. *ACM Signum Newsletter*, 20(4):2–5, 1985.

Akhil Gudipaty

Academic Vitae

EDUCATION

The Pennsylvania State University, University Park, PA May 2023
Schreyer Honors College
Bachelor of Science in Computer Science
Minor: Statistics

SKILLS AND COURSES

- **Courses**: Operating Systems, Systems Programming, Artificial Intelligence, Data Structures and Algorithms, Digital Design, Computer Organization and Design, Object Oriented Programming, Linear Regression, Database Management Systems, Application Programming, Theory Of Computation, Programming Language Concepts
- **Skills**: Python, Java, C, C++, Swift, HTML, CSS, Django, Flask, SQL, Relational Databases, Git, AWS, Cloud Computing, Knowledge on Agile Scrum Methodology, Knowledge on DevOps pipelines and processes, OS Knowledge: MacOS, Windows, Linux(Ubuntu and Redhat)

EXPERIENCES

Ernst & Young **San Francisco, CA**
Cybersecurity Consulting Intern June 2022 – August 2022

- Worked on multiple client engagements that include developing and delivering technology strategies, architectures and roadmaps for clients to enable their business strategies.
- Our team managed large-scale technology-enabled business transformations as well as advised clients on areas such as system selection and vendor selection, helping them **optimize their IT costs by almost 30%** and establish effective IT governance.
- Worked in **Cloud Security and Penetration testing** cybersecurity fields.

National Aeronautics and Space Administration (NASA) **Remote**
Software Engineering Intern August 2021 – May 2022

- Designed and developed a new, interactive, web-based tool to improve the use of lessons learned across ground and flight projects related to NASA missions. Increased **efficiency and execution time by 50%** from the previous version.
- Used web technologies like **Django, HTML, CSS, JavaScript, and Docker**.
- Developed this project using Agile/Scrum processes including sprint meetings, daily standup meetings, and sprint reviews.

RELATED EXPERIENCES

Research Assistant **University Park, PA**
Pennsylvania State University September 2022 – Present

- The goal of the research is to obtain results from cyclic POMDP algorithms and convert that into a system of equations involving circulant matrices, which can be solved using enhanced circulant matrices solvers
- Used Python to code these enhanced circulant matrices solvers

PROJECTS

PartyFinder **University Park, PA**
Founder May 2022 – Present

- Designed and developed an industry level application that allows college students to find parties and other social events near them. Used **Swift** and **Firebase(Database)** to launch this app

HEALER (AI Project) **University Park, PA**
Project Leader January 2021 – May 2021

- Implemented a software agent using **C++,** which is a POMDP problem, that recommends sequential intervention plans for the homeless shelters, who organize these events to raise awareness about HIV among homeless youth

HACKPSU (Hackathon) **University Park, PA**
Project Leader August 2021 – August 2021

- Developed a website that can assess the mental health or state of mind of a person. Developed using **Python Django** on the backend and **HTML**, **CSS** on the front end