PRUNING ALGORITHMS FOR PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELİM ÖZGEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

NOVEMBER 2017

Approval of the thesis:

PRUNING ALGORITHMS FOR PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

submitted by **SELİM ÖZGEN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Tolga Çiloğlu
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Mübeccel Demirekler
Supervisor, **Electrical and Electronics Eng. Dept., METU** _____

Examining Committee Members:

Doç. Dr. Umut Orguner
Electrical and Electronics Eng. Dep., METU _____

Prof. Dr. Mübeccel Demirekler
Electrical and Electronics Eng. Dep., METU _____

Prof. Dr. Faruk Polat
Computer Eng. Dep., METU _____

Prof. Dr. Ömer Morgül
Electrical Eng. Dep., Bilkent University _____

Assist. Prof. Dr. Mehmet Tan
Computer Eng. Dep., TOBB ETU _____

Date: **30.11.2017**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SELİM ÖZGEN

Signature :

ABSTRACT

PRUNING ALGORITHMS FOR PARTIALLY OBSERVABLE MARKOV DECISION PROCESSES

Özgen, Selim
Ph.D., Department of Electrical and Electronics Engineering
Supervisor : Prof. Dr. Mübeccel Demirekler

November 2017, 120 pages

It is possible to represent the value function in partially observable Markov decision processes as a piecewise linear function if the state, action, and observation space is discrete. Exact value iteration algorithm searches for this value function by creating an exponential number of linear functions at each step, many of which can be pruned without changing the value of the value function. The pruning procedure is made possible by the use of linear programming.

This study first gives a geometric framework of the pruning procedure. It shows that the linear programming iterations refer to the selection of different convex regions in the vector space representation of the pruning problem. We also put forward an algebraic framework, which is the utilization and maintenance of linear programs. It shows how the problem can be decomposed into small sized LPs and what the LP iterations refer to. While stating these two theoretical frameworks, their relations have also been exploited.

The exponential increase in the number of vectors in any step of the exact value iteration algorithm is due to an operation called the cross-sum addition of a set of vectors. This operation results in a new set of vectors. It is known that for any of the *summand* vectors in this new set to be non-dominated, the *addend* vectors entering the cross-sum addition should have intersecting support sets. The given geometric and algebraic framework has further been extended to exploit this particular property of the cross-sum operation.

Two novel pruning algorithms have been offered in this study. First algorithm, called *FastCone*, can be used for pruning any given set of vectors. For a given set of clean vectors at any step, the algorithm hastily searches for the convex region that a dirty vector is in and tries to find a clean vector if only the given set of clean vectors is not sufficient to make the decision about this dirty vector.

The second algorithm is called *Cross-Sum Pruning with Multiple Objective Functions*, where the aim is to find the vectors that have non-intersecting support sets with the current active vectors in each simplex iteration. This approach is useful because when two vectors from two different sets with non-intersecting support sets are detected, it is possible to delete all ordered pairs containing these two vectors. And this amounts to a simple sign check of the coefficients of a row of the simplex tableau.

To show the algorithms' performance, both algorithms have been compared to the conventional algorithms and their revised versions both analytically and experimentally.

Keywords: decision-theoretic planning, Markov decision processes, partial observability, linear programming

ÖZ

KISMİ GÖZLEMLENEBİLİR MARKOV KARAR SÜREÇLERİ İÇİN BUDAMA ALGORİTMALARI

Özgen, Selim
Doktora, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Mübeccel Demirekler

Kasım 2017 , 120 sayfa

Durum, eylem ve gözlem uzayının ayrık olduğu kısmi gözlemlenebilir Markov karar süreçlerinde değer fonksiyonunu parçalı doğrusal bir fonksiyon olarak göstermek mümkündür. Kesin değer yineleme algoritması, bu değer fonksiyonunu ararken her adımda üssel sayıda lineer fonksiyon yaratmaktadır. Bu fonksiyonların önemli bir kısmını değer fonksiyonunun değerini hiç değiştirmeden elemek mümkündür. Bu budama prosedürü lineer programlamanın kullanılması sayesinde mümkün olmaktadır.

Bu çalışma ilk olarak budama prosedürünün geometrik bir çerçevesini vermektedir. Bu çalışmada gösterilmektedir ki, lineer programlama iterasyonları, budama probleminin vektör uzayı gösteriminde farklı dışbükey alanların seçimine denk gelmektedir. Buna ek olarak, budama problemine cebirsel bir çerçeve de sunulmuştur. Bu çerçeve lineer programların inşa edilmesi ve kullanılması üzerine kurulmaktadır. Problemin daha küçük boyutlu lineer programlar kullanılarak

nasıl çözülebileceği ve lineer programların iterasyonlarının ne anlama geldiği anlatılmıştır. Problemin geometrik ve cebirsel çerçevesi arasında ayrıca bir ilişki de kurulmuştur.

Kesin değer yineleme algoritmasının her adımında vektör sayısındaki üssel artışın nedeni verili olan vektör kümeleri üzerinde yapılan çapraz toplama işlemidir. Bu işlem sonucunda yeni bir vektör kümesi oluşmaktadır. Bilinmektedir ki, yeni oluşan setteki toplanan vektörlerden herhangi birinin elenebilir olduğunu görmek için çapraz toplama işlemine giren toplanan vektörlerin destek kümelerinin kesişimine bakmak yeterlidir. Elinizdeki çalışma, verili olan geometrik ve cebirsel çerçeveyi çapraz toplama operasyonunun özelliklerini incelemek üzere kullanmaktadır.

Bu çalışmada iki yeni budama algoritması önerilmektedir. Bunlardan ilki olan *FastCone* verili herhangi bir vektör seti için kullanılabilir. Algoritmanın herhangi bir anında verili olan bir temiz vektör seti için, seçilmiş olan kirli vektörün içine düştüğü dışbükey alan hızlı bir şekilde bulunmaktadır. Eğer bulunan çözüm, seçilmiş olan kirli vektörü elemek için yeterli değilse bu işlem için yararlı olabilecek temiz vektörler bulunmaya çalışılmaktadır.

İkinci algoritmanın ismi *Cross-Sum Pruning with Multiple Objective Functions* olarak belirlenmiştir. Bu algoritma ile amaçlanan herhangi bir simpleks adımında aktif olan vektörlerin destek kümeleriyle kesişimi boş küme olan vektörleri belirlemektir. Bu operasyonun işlevi şöyle özetlenebilir. Eğer farklı iki kümeden alınan iki vektörün destek kümelerinin kesişimi boş küme ise, bu iki vektörü içeren bütün sıralı çiftlerin elenmesi mümkün hale gelmektedir. Bu iki vektörün destek kümelerinin kesişiminin boş küme olduğunu anlamak için ise simpleks tablosundaki bir sırada işaret kontrolu yapmak yeterli olmaktadır.

Algoritma performanslarını gösterebilmek için önerilen algoritmalar, konvansiyonel algoritmalar ve onların revize edilmiş versiyonları ile analitik ve deneysel olarak kıyaslanmıştır.

Anahtar Kelimeler: karar kuramı temelli planlama, Markov karar süreçleri,

kısmi gözlemlenebilirlik, lineer programlama

To my grandmother Nezaket Erigür

ACKNOWLEDGMENTS

I had a really long career in higher education and had a valuable lesson. It is more about *whom* you work with than *what subject* you work about. After a great deal of suffering on my part, I was lucky enough to find a mentor who has the gifts of active empathy, patience, and discipline. I would sincerely like to thank my *Doktormutter* Mübeccel Demirekler. I still have a lot to learn from her.

I sincerely thank the other members of my thesis monitoring committee; Umut Orguner and Faruk Polat. They were supportive at every step of this thesis and shared my excitement on the subject. I would also like to thank Ömer Morgül and Mehmet Tan for their valuable comments in my thesis jury.

I have learnt a lot from many professors in my institute but I would like to especially thank Emre Özkan, Emre Tuna and Arzu Koç. Their sincerity always reminded me that our relation was not restricted to a few semesters of lectures.

I want to thank Elif Sarıtaş, Murat Kumru, Cumhur Çakmak, Mehmet Mutlu, Hasan İhsan Turhan, Oktay Sipahigil, Mehmet Çetinkaya, Ahmet Musab Elbir for the academic cooperation we had in the department over the years. I should mention Erkin Çilden for his support on any subject on POMDPs. It is not possible to list all good friends from my METU years, but I would like to mention the names of Özgür Sarı, Caner Ünal, Gökçe Oğuz, Zelha Nil and Azadeh Kamali Tafreshi. I would also like to thank my volleyball group. I guess we were the craziest ones in the *Devrim* Stadium.

I have a friendship that goes on for decades with Taylan Eren Yenilmez, Özgür Burçak Gürsoy, İmge Yıldırım and Başak Deniz Özdoğan. They have always been one phone call away and I am truly thankful to all of them. It has been less than a decade since I sat near Mürsel Karadaş at a lecture on optimization

at METU and he introduced me to Pınar Şen and Tuğcan Aktaş. We now form a group which proved to be quite resilient over the time. I get to know Yusuf Barış Güleç for even a smaller amount of time, yet his support has proven to be solid as a rock. I cherish his trueness and compassion. I am happy to have such lifetime companions.

Many members of my extended family should be mentioned for their warmth and support, but I would like to use this chance to express gratitude to my nuclear family. My sister Elif Yeşim Özgen Kösten has never led me to think that the age gap between us was something that would impede a sincere discussion. She is my youthful older sister. My mother, Gülseren Özgen, has thought every step of my journey with (and many times ahead of) me. This is a gift that people value more as they get older. My father, İsmet Tamerkan Özgen, has always been respectful and supportive of the decisions that I have taken. Albeit the huge differences in their personalities, these two people have formed an harbor that I can safely take shelter in in any case of emergency. They are my rocks.

This research was supported with a PhD scholarship by the Scientific and Technical Research Council of Turkey (TÜBITAK).

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
LP	Linear Programming
PWLC	Piecewise Linear Continuous
LEX	Lexicographic Ordering
LRK	Lark's Algorithm
LRwI	Lark's Algorithm with Initial Condition
ISKY	Iterative Skyline Algorithm
ISwM	Iterative Skyline Algorithm with Multiple Objective Functions
FC	FastCone Algorithm
IP	Incremental Pruning Algorithm
GIP	Generalized Incremental Pruning Algorithm
IBIP	Intersection Based Incremental Pruning
RBIP	Region Based Incremental Pruning
CSwM	Cross-Sum Pruning with Multiple Objective Functions

CHAPTER 1

INTRODUCTION

Decision making under uncertainty has been a subject of interest since the 1960s [1–4]. The milestone for the research has undoubtedly been the invention of dynamic programming in the 1950s [5–8]. Dynamic programming was theoretically exciting, but at the same time it was computationally expensive. Due to the lack of computational capacity, the application areas remained restricted until the 1980s. As the inventor of the algorithm, Richard Bellman notes [9], "I was prepared to face up to the challenge of using dynamic programming as an effective tool for obtaining numerical answers to numerical questions. A considerable part of the motivation in this direction at that time was the continuing development of the digital computer... It is far more difficult to obtain an effective algorithm than one that stops with a demonstration of validity." Yet, dynamic programming has proved to be much more than its theoretical rigor and many algorithms exploiting this framework have been developed to come up with exact and approximate solutions to the planning problem.

This thesis is a contribution to the exact representation of the value function in finite horizon for partially observable Markov decision process. A partially observable Markov decision process (POMDP) models an agent acting in an uncertain environment with imperfect actuators and noisy sensors. Due to imperfect actuators, the effect of the action might not be deterministic and that brings forward the use of a probabilistic model for the state transition with respect to the selected action. Moreover, the noise in sensors causes the state to be observed partially; thus we need to define a probabilistic relation for receiving

an observation when any state and action couple is given. Note that although the state transitions and observation probabilities are defined with respect to the action taken, the model is not complete as there is no incentive for the agent to act. This incentive is given in the form of a reward function; the agent is given a reward for acting in a particular way depending on its state. For a given planning horizon, this reward model is used to calculate the cumulative reward for any given sequence of actions. This cumulative reward is called the value function.

Above framework is eligible for many real life problems that require complex models. Therefore POMDPs have received much attention and applied in diverse areas such as preference elicitation for different uses of an intelligent system [10], dialogue management [11], radar resource management [12], scheduling in sensor networks [13], healthcare [14–17], target tracking [18], collision avoidance [19], etc. Even though POMDPs provide the possibility for modeling various phenomena, the huge computational cost for coming up with an exact solution limits its use.

The value function of a POMDP can be calculated by dynamic programming. Moreover, when the model has discrete state, action, and observation spaces, the value function comes out as a piecewise linear convex function for any finite planning horizon [20]. A piecewise linear convex function can be represented by a set of vectors where each vector is used to represent the value function for some convex subset of the solution set. However during the dynamic programming update, the number of possible vectors grows linearly with the cardinality of the action set and exponentially with the cardinality of the observation set [21]. Even in the finite horizon planning case, finding the optimal policy which maps the belief states to actions is a PSPACE-hard problem [22].

Improving the scalability of solution methods for POMDPs has been a critical research topic since the 1970s [23–25], but the attention reached a peak at the beginning of 2000s. This is due to point based value iteration algorithms, where the value function computations are done only for a finite subset of the belief space [26–28]. For the infinite horizon problem with a discount factor, the value

iteration algorithm is a contraction mapping on bounded real functions [29]. Many of the point based algorithms exploit the properties of the error bound for this contraction mapping [30–32]. There are also approximate solutions of the optimal planning problem by policy iteration where the policy search has been realized in a set of restricted policy space [33, 34]. In any of these algorithms, a number of iterations is necessary for attaining to a predetermined bound on the error of the value function [35]. Yet for the finite horizon case, it is not easy to give an estimate of the error bound between the optimal and approximate value functions. It is still possible to use heuristic solvers, but we can safely say that there is a trade-off between accuracy and speed of the solution. When high accuracy of the solution is required, the use of exact value iteration algorithms is inevitable.

The computation of all possible vectors for each step of the exact value iteration algorithm is quite straightforward. But this set of vectors should be pruned to a minimal subset retaining the same value function over the state space. The task of removing the useless vectors is typically known as *pruning* and is done by solving a number of linear programs (LPs). The number of linear programs to be solved for pruning a set of vectors is directly related to the initial number of vectors in the set. With an exponential increase in the number of vectors, the number of LPs dramatically increases even for a small planning horizon. Thus, most of the time in the dynamic programming update is spent for solving these LPs.

To deal with this bottleneck, this thesis focuses on the particular structure of the linear programs to be solved in the pruning operation [24,28,36,37]. In this work, we give a geometric framework of the pruning procedure by using the vector state representation of the value function. The dual representation of the value function of finite horizon POMDPs in belief set and vector space has been noted by Zhang [38]. We demonstrate what primal and dual simplex iterations mean for any given set of vectors in the vector space representation of the problem. We show that the linear programming iterations refer to the selection of different convex regions in the vector space representation of the pruning problem. All steps of this problem is discussed in tandem with an algebraic framework which

has also been explained in great detail. By the algebraic framework we refer to the construction and utilization of linear programs. The exhaustive explanation of the relation between the algebraic and geometric frameworks is a contribution of this thesis. This study shows how the problem can be decomposed into small sized LPs and what each LP iteration refers to.

In any step of the exact value iteration algorithm, the input vectors are first multiplied by different projection matrices resulting in multiple set of vectors. After this operation, the Cartesian product of these new set of vectors are taken. Each n-tuple refers to a different selection of vectors and the vectors in an n-tuple are summed up to find an action dependent value function vector. This is called as the cross-sum addition [1] of a set of vectors and there is an exponential increase in the number of vectors is due to the cross-sum addition. There is a field of research that exploits the properties of the dynamic programming update steps to decrease the complexity of the LPs to be solved, many of which attack the special structure of the cross-sum operation [41–45]. The given geometric and algebraic framework has further been extended to exploit this particular property of the cross-sum operation.

Finally, we offer two novel pruning algorithms based on the theoretical framework presented in this study. The first algorithm is called FastCone. For a given set of clean vectors, the algorithm quickly searches for the convex region that a dirty vector is in and searches for another clean vector if only the current set of clean vectors is not sufficient to prune this dirty vector. The second algorithm is called Cross-Sum Pruning with Multiple Objective Functions, where the aim is to find the vectors that have non-intersecting support sets with the current active vectors in each simplex iteration. Due to the properties of the cross-sum operation, vector elimination is performed without explicitly writing all of the dirty vectors to the simplex tableau.

All codes of the existing and novel algorithms are written in MATLAB environment. In any of the algorithms presented, Bland's rule has been used [46] for the

[1] This operation is well known in convex analysis and defined as Minkowski addition [39]. The term was named after the founder Hermann Minkowski [40]. We will stick to the term cross-sum addition as preferred in the decision theoretic planning community.

simplex iterations and the degeneracy of the simplex iterations has been solved by the Harris ratio test [47]. The novel algorithms are compared to the existing algorithms both analytically and experimentally.

This thesis is structured as follows. Chapter 2 gives a rigorous overview on the decision-theoretic planning for the discrete state, action, and observation space. This chapter discusses the properties of the value function for POMDPs and gives an upper-bound complexity result for the calculation of the exact value function for a given planning horizon. Chapter 3 discusses the pruning problem for where an arbitrary set of vectors are reduced to a minimal set where each vector is maximal at some belief state. Known pruning algorithms are discussed and the pruning problem is explained in an algebraic and geometric framework. Chapter 4 exploits the theoretical framework for the cross-sum operation where the number of vectors increase exponentially. Chapter 5 concludes this study.

CHAPTER 2

DECISION-THEORETIC PLANNING UNDER UNCERTAINTY

2.1 Introduction

Decision-theoretic planning means deciding on a sequence of actions for an agent in an environment to complete a task. Two important concepts will be described here. Agent and its interaction with its environment is called the *system*. The information that is necessary for deciding on the sequence of actions is called as the *state*. The sequence of actions taken is called the *decision process* and finding the sequence of actions that are optimal in some sense is called the *decision theory*.

While there is uncertainty in the system, the future behavior is not completely unforeseeable by looking at its present state and future control actions, as in a deterministic system. The uncertainty in the system can be due to two different reasons: there can be an uncertainty about the consequences of the actions taken or there can be an uncertainty about what we observe about the state. The property of the uncertainty completely changes the approach to the problem.

In this thesis, we will deal with sequential decision problems in a discrete time framework. That means that at every discrete time step, a decision about the system should be made and this decision affects the system state in the following time steps. The number of decisions to be made can be finite or infinite regarding to the number of time steps taken into consideration [20, 44, 48]. Both of these problems have been thoroughly dealt with. Yet, our focus will be on finite

horizon problems and the infinite horizon case will be mostly explained for the sake of completeness.

2.2 Markov Decision Processes

Markov Decision Process (MDP) is a useful tool for sequential decision making in a stochastic environment [49]. MDP actually refers to completely observable MDP where the state of the agent can be directly observed. Yet, completely knowing the state does not make the system deterministic. What makes it different from a deterministic system is that the agent is not sure about the consequences of the action she takes. However, once the action is taken, the state of the agent at the following discrete time step can also be directly observed.

How does the Markovian property come into the picture? The Markovian property asserts that knowing the current state information at any point in time is enough to act optimally. When the state transition is Markovian, the past states and actions become irrelevant to the estimation of the future states once the current state is known.

After giving a verbal description of the problem, we will now depict it formally. For this, we need to first define the support set of the variables used. In mathematical terms, MDP is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$, where;

- $\mathcal{S}$ corresponds to a finite set of world states where each state will be denoted by $s \in \mathcal{S}$
- $\mathcal{A}$ is a finite set of actions that the agent can execute where each action will be denoted by $a \in \mathcal{A}$
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines the transition probability distribution $p(s'|s, a)$ that describes the effect of action a on the state of the world. $s' \in \mathcal{S}$ is a random variable that described the state after action a is taken when state was s. This transition function models the stochastic nature of the environment.
- $r(s, a) \in \mathcal{R}$ corresponds to the reward models $\mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ that the agent receives for executing action a when the state is s.

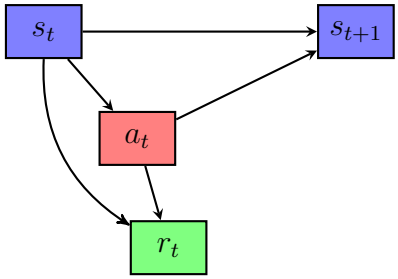


Figure 2.1: MDP Influence Diagram

MDP assumes that at any time step t, the system is at a state s and the agent takes an action a and receives a deterministic reward $r(s, a)$ from the environment as a result of this action. The system switches to a new state s' according to a known stochastic model $p(s'|s, a)$. Due to this uncertainty in the transitions, the states in the future time steps cannot be known exactly at time t.

While we search for the the best action we can take, we should also use our information about the future states to decide on the future actions. We will define the variables s_t, a_t as the state and action at the time step t, respectively. The values of these variables are not known before time t. When the sequence of actions are known, s_t becomes a random variable defined over $\mathcal{S}$. Note that the agent can decide on her decisions *in advance* and apply this strategy no matter what the state is. In such a case, there won't be any ambiguity about which action is selected. However, if the agent decides on selecting her future actions with respect to the future values of the states, an uncertainty about the actions arise. In this case, a_t becomes a function of the future state and has a random distribution over $\mathcal{A}$. As any function of a random variable is also a random variable, $r_t := r(s_t, a_t)$ also becomes a random variable distributed over $\mathcal{R}$. The influence diagram for a Markov decision process can be seen in Figure 2.1.

We would like to make a clarification here. The transition probability distribution $p(s^{'}|s, a)$ and the reward model $r(s, a)$ can also be changing with time; and in such a case, we would need to define $p_t(s^{'}|s, a)$ and $r_t(s, a)$. Such an attempt would be necessary if the agent's interaction with the environment is changing with time. Note that the Markovian property would be preserved in this case. In our case, the system model is *stationary*. Therefore $r_t = r(s_t, a_t)$.

The Markov property entails that the random variable s_{t+1} is only dependent on the distribution of the previous state s_t and action a_t. In mathematical terms,

$$p(s_{t+1}|s_t, \ldots, s_0, a_t, \ldots, a_0) = p(s_{t+1}|s_t, a_t)$$

As there are a finite number of states, define $D := |\mathcal{S}|$. Then, this probability distribution can be described by a matrix $\mathbf{P}(a) \in \mathbb{R}_{\geq 0}^{D \times D}$, as follows:

$$(\mathbf{P}(a))_{ij} := p(s_{t+1} = j|s_t = i, a_t = a) \tag{2.1}$$

Similarly, we define the immediate reward function as a vector $\mathbf{r}(a) \in \mathcal{R}^{D \times 1}$, as follows:

$$(\mathbf{r}(a))_i := r(s_t = i, a_t = a) \tag{2.2}$$

2.2.1 Optimality Criteria

How do we define the best action to take at time t? For the sake of simplicity we will assume that $t = 0$. As the state is completely observable, the value of s_0 is known. But, as we have seen from the previous section, the future state and actions can only be known probabilistically. Therefore our aim would be to find the sequence of decisions that would maximize some form of long term reward. Define this as a function of the rewards $J(r_0, \ldots, r_N)$ where N is the planning horizon. This function would clearly be a stochastic function.

One reasonable candidate for the performance measure then would be the expectation of the sum of rewards:

$$J(r_0, \ldots, r_N; s_0) = E_{R^N}\left(\sum_{t=0}^{N} r_t \middle| s_0\right) = E_{S^N}\left(r(s_N) + \sum_{t=0}^{N-1} r(s_t, a_t) \middle| s_0\right) \tag{2.3}$$

where $E(.)$ is the expectation operator, $R^N = r_{0:N}$, $S^N = s_{0:N}$. The important observation here is that the only random quantity in this expectation is the states. The actions become random when they are described as functions of the states.

As mentioned before, our aim is to find the best sequence of actions $\{a_t\}_{t=0}^{N-1}$ that would maximize the reward function $J(.)$. The strategy used for selecting

an action is called a *policy*. The policy decision can be made without considering the system state and this is called an open loop policy. In this case, the sequence of actions can be determined at time $t = 0$ as our decisions are not dependent on the value of the future states. On the other hand, we can use the information about the system state for deciding on the action at each time step t. This is called a closed loop policy. Clearly, using a closed loop policy can alleviate the uncertainty inherent to the system as it considers the state of the agent while executing the action at each time step. However, in this case, it is not possible determine a sequence of actions to be executed at time $t = 0$, as we would prefer to see the state at any time to decide on the preferable action. However, it is still possible to define a mapping from the states to the actions for each time step $t = 1, \ldots, N$ when we are at time $t = 0$. As the policy can be evaluated for each time step at $t = 0$, it can be applied as soon as the states at the next time steps become available. The evaluation of this closed loop policy is what we refer as planning. Moreover as discussed in [50], an open loop policy is actually a degenerate case of closed loop policy.

Therefore, a closed loop policy is a mapping from the support set of the states to the actions. In mathematical terms, it can be defined as $\mu(s) : \mathcal{S} \to \mathcal{A}$. If the policy definition changes with time, then it becomes μ_t. For a fixed planning horizon N, we need to also define a *plan*, which is a sequence of policies for each time step $0 \leq t < N$. In mathematical terms, a plan is $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\} : \mathcal{S} \times \mathcal{S} \ldots \times \mathcal{S} \to \mathcal{A} \times \mathcal{A} \ldots \times \mathcal{A}$.

When we decide on a closed loop plan π, the expected cost 2.3 becomes

$$J^{\pi}(s_0) = E\left(\sum_t r(s_t, \mu_t(s_t))\middle| s_0\right) \tag{2.4}$$

Here $E(.)$ denotes the expectation with respect to probability distribution of $\{s_t\}$ and $\{a_t\}$ determined by μ_t. Note that, when the plan π is fixed, this expectation can be calculated if the value of the initial state, s_0 is known.

Recall from 2.1 that for a given action a, the transition probability distribution can be represented in the form of a matrix $\mathbf{P}(a)$. When a policy μ is fixed, it is

also possible to define the vector $\mathbf{r}(\mu) \in \mathcal{R}^{D\times 1}$ and matrix $\mathbf{P}(\mu) \in \mathbb{R}_{\geq 0}^{D\times D}$:

$$(\mathbf{r}(\mu))_j = r(j, \mu(j)) \tag{2.5}$$

$$(\mathbf{P}(\mu))_{ij} = \mathbf{P}(\mu(i))_{ij} \tag{2.6}$$

Define $\mathbf{e_i} \in \mathbb{R}_{\geq 0}^{1\times D},\ \sum_j \mathbf{e_i}(j) = 1$ such that $\mathbf{e_i}(i) = 1$. Assume that at $t = 0$ it is known that $s_0 = i$. Then the prior distribution of the state is $p_0 = \mathbf{e_i}$.

$$J^\pi(i) = \sum_{t=0}^{N}\sum_{j=1}^{D} p(s_t = j | s_0 = i, a_0 = \mu_0(i)) r(j, \mu_t(j)) \tag{2.7}$$

$$J^\pi(i) = \sum_{t=0}^{N} \mathbf{e_i}(\mathbf{P}(\mu_0)\dots\mathbf{P}(\mu_t))\mathbf{r}(\mu_t) \tag{2.8}$$

Thus, the best Markov plan π^* would be the one maximizing this reward function. Note that for an open loop control sequence independent of the states, the calculation of Equation 2.8 would be trivial because $\mathbf{P}(\mu_k) = \mathbf{P}(a_k)$ for some fixed value of $a_k \in \mathcal{A}$. With a closed loop Markov plan, this problem becomes nontrivial and can be solved by dynamic programming [6].

2.2.2 Dynamic Programming

Dynamic programming is a technique for calculating the reward of a Markov plan, π [7]. The technique depends only on the fact that the state process corresponding to π is Markov. That is to say, for any fixed plan π and any time step t if $s_t = i$ is given, the calculation of the expected cost for the future time steps can be done independent of the past states of the system.

Define $V_t^\pi(i)$ for some fixed plan $\pi = \{\mu_0, \mu_1, \dots, \mu_{N-1}\}$ as:

$$V_t^\pi(i) = E\left(r(s_N) + \sum_{k=t}^{N} r(s_k, \mu_k(s_k)) \middle| s_t = i \right) \tag{2.9}$$

$$V_t^\pi(i) = \sum_{k=t}^{N} \mathbf{e_i}(\mathbf{P}(\mu_t)\dots\mathbf{P}(\mu_k))\mathbf{r}(\mu_k) \tag{2.10}$$

where $\mathbf{e_i} \in \mathbb{R}_{\geq 0}^{1\times D},\ \sum_j \mathbf{e_i}(j) = 1$ and $\mathbf{e_i}(i) = 1$.

Now, define the column vector $(\mathbf{V}_t^\pi)_i = V_t^\pi(i)$. Then it is possible to write the

following the backward recursion formula for all $0 \leq t < N$,

$$\mathbf{V}_t^{\pi} = \mathbf{r}(\mu_t) + \mathbf{P}(\mu_t)\mathbf{V}_{t+1}^{\pi} \tag{2.11}$$

It can be seen that

$$J^{\pi}(s_0) = \sum_{i=1}^{D} p(s_0 = i)(\mathbf{V}_0^{\pi})_i \tag{2.12}$$

starting with the final condition $(\mathbf{V}_N^{\pi})_i = r(i, \mu_N(i))$.

It is shown in [8, 50] that for the optimal policy π^*, we can write

$$(\mathbf{V}_t^{\pi^*})_i = \sup_{a \in \mathcal{A}} \mathbf{e_i} \left(\mathbf{r}(a) + \mathbf{P}(a)\mathbf{V}_{t+1}^{\pi^*}\right) \tag{2.13}$$

where the supremum is calculated separately for each component of the column vector $\mathbf{V}_t^{\pi^*}$. Assume that for $s_t = i$, the maximum value for $(\mathbf{V}_t^{\pi^*})_i$ is given by $a_t = k$. Then $\mu_t^*(i) = k$. Moreover,

$$J^{\pi^*}(s_0) = \sum_{i=1}^{D} p(s_0 = i)(\mathbf{V}_0^{\pi^*})_i$$

When the planning horizon $N = \infty$, a stationary and deterministic policy is available. For this case, policy and plan can be used interchangeably while the optimal plan becomes the application of the optimal policy at every time step [49, 50]. As we will deal with the finite horizon case in this thesis, we will not go into the details of this derivation.

2.3 Partially Observable Markov Decision Processes

The main distinction between MDPs and POMDPs is in the information one uses to select an action. For the MDP case, as shown in Equation 2.13, the policy μ_t^* is calculated by fixing $s_t = i$ and calculating $\mathbf{V}_t^{\pi}$ for all possible values of $a_t = a, a \in \mathcal{A}$. As both $\mathcal{S}$ and $\mathcal{A}$ are finite, this is a viable operation. This assumption is possible because at time t, the process state s_t will be known with certainty.

For POMDP, actions are based only on the available information that consists of previous observations and actions. Observations correspond to features of

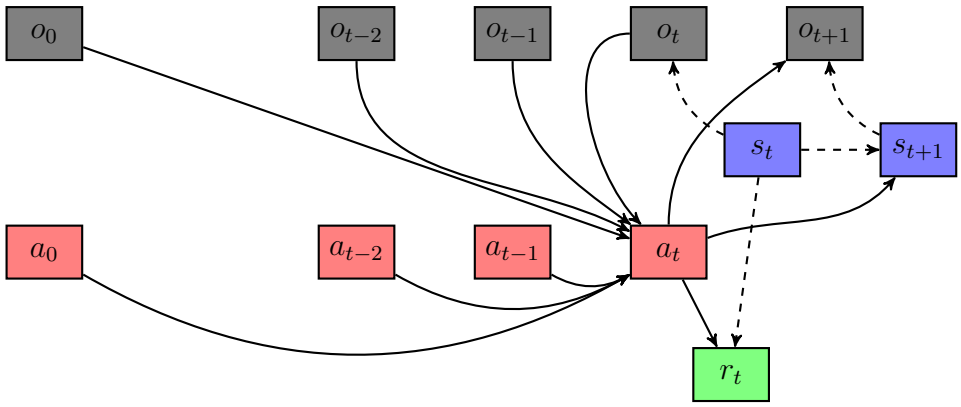


Figure 2.2: POMDP Influence Diagram with an emphasis on incoming and outgoing branches to variable a_t. To stress that the state is a hidden variable, the branches related to the states are shown with dashed lines.

the world directly perceptible by an agent's sensors. In the case of partial observability, the mathematical definition becomes $(\mathcal{S}, \mathcal{A}, \Theta, \mathcal{T}, \mathcal{O}, \mathcal{R})$, where;

- Θ is a finite set of observations where each observation will be denoted by $o \in \Theta$
- $\mathcal{O} : \Theta \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines the observation probability distribution $p(o|s, a)$ that models the effect of actions and states on observations

Adding to our discussion on MDPs, we will also define the random variable o_t, which is the observation at time t. Obviously, the value of o_t is not known before time t.

The difference between the two models can be understood by comparing Figure 2.1 and 2.2. In Figure 2.1, the action is taken directly by knowing the state we are in at each time step. In Figure 2.2, we can see that as the states are not completely observable by the decision agent, all observations and past decisions are used to estimate the state and the action is taken according to all information in hand.

Optimal behavior in a POMDP requires access to the entire history of the process. This statistic is known as an *information state* or *belief state*. An information state represents all information available to the agent at the decision time that is relevant for the selection of the optimal action.

In mathematical terms, define $I^t := (o_0, \ldots, o_t, a_0, \ldots, a_{t-1})$. Note that, I^t has all the information available to the user at time t. We are interested in finding the state distribution at time t, therefore we define $\mathcal{B} = \Pi(\mathcal{S})$ to be the space of all probability distributions over $\mathcal{S}$. Then, $\mathcal{B}$ is a unit-D simplex.

Definition 1. *Unit-D simplex is defined as*

$$\mathcal{B} = \{b \in \mathbb{R}_{\geq 0}^{1 \times D} : \quad b\mathbf{e} = 1\} \tag{2.14}$$

where $\mathbf{e} = [1 \ldots 1]^T$.

As can be seen from Definition 1, unit-D simplex is set in a $D-1$ dimensional space.

Definition 2. *The probability distribution over* $\mathcal{S}$ *at any time* t *can be defined as*

$$b_t(i) = p(s_t = i | I_t), \ b_t \in \mathcal{B}, 1 \leq i \leq D \tag{2.15}$$

Keeping in mind that $I^{t+1} = (I^t, o_{t+1}, a_t)$, and using the Bayes' rule [20],

$$\begin{aligned}
b_{t+1}(i) &= p(s_{t+1} = i | I^{t+1}) && (2.16)\\
&= p(s_{t+1} = i | I^t, o_{t+1} = o, a_t = a) && (2.17)\\
&= \frac{p(s_{t+1} = i, o_{t+1} = o | I^t, a_t = a)}{p(o_{t+1} = o | I^t, a_t = a)} && (2.18)\\
&= \frac{\sum_j p(s_{t+1} = i, o_{t+1} = o | s_t = j, I^t, a_t = a) p(s_t = j | I^t, a_t = a)}{p(o_{t+1} = o | I^t, a_t = a)} && (2.19)\\
&= \frac{\sum_j p(o_{t+1} = o | s_{t+1} = i, s_t = j, I^t, a_t = a) p(s_{t+1} = i | s_t = j, I^t, a_t = a) b_t(j)}{p(o_{t+1} = o | I^t, a_t = a)} && (2.20)\\
&= \frac{p(o_{t+1} = o | s_{t+1} = i, a_t = a) \sum_j p(s_{t+1} = i | s_t = j, a_t = a) b_t(j)}{p(o_{t+1} = o | I^t, a_t = a)} && (2.21)
\end{aligned}$$

We can see that, the denominator of Equation 2.21 is actually a normalization factor. We have also written Equation 2.17 explicitly to stress that the observation o_{t+1} and the action a_t are already known by the agent at time $t+1$.

Using Equation 2.1 and defining the diagonal matrix $(\mathbf{D}(a, o)) \in \mathbb{R}_{\geq 0}^{D \times D}$:

$$(\mathbf{D}(a, o))_{ii} := p(o_{t+1} = o | s_t = i, a_t = a), \ 1 \leq i \leq D \tag{2.22}$$

$$(\mathbf{D}(a, o))_{ij} := 0, \ 1 \leq i, j \leq D, i \neq j \tag{2.23}$$

we arrive at

$$b_{t+1} = \frac{b_t \mathbf{P}(a)\mathbf{D}(a,o)}{b_t \mathbf{P}(a)\mathbf{D}(a,o)\mathbf{e}} \tag{2.24}$$

where $a_t = a$, $o_{t+1} = o$ and $\mathbf{e}$ is defined before.

It means that when we have the belief vector b_t at time t and make a decision $a_t = a$ and observe $o_{t+1} = o$ according to our decision, we can update our belief vector to b_{t+1}. There is no more information available at any of the past observations and actions while deriving b_{t+1} when b_t is known. Due to this reason the belief state, b_t is called the sufficient statistics for POMDPs. By the aid of the belief state, POMDPs can be represented as belief-state MDPs; thus allowing the use of the properties of MDPs. The equivalence of information state and belief state representation of the problem is shown in Figure 2.3.

2.3.1 Optimality Criteria

The performance measure can be taken similar to Equation 2.3:

$$J(b_0) = E_{R^N}\left(\sum_{t=0}^{N} r_t \middle| b_0\right) = E_{S^N}\left(r(s_N) + \sum_{t=0}^{N-1} r(s_t, a_t) \middle| b_0\right) \tag{2.25}$$

As in the MDP case, we are searching for the best actions to take to maximize this reward function. We need to define a policy μ_t for each time step. If the states were available, we would be able to define this policy by considering the states. As the states are not completely observable, we need to use the information state I^t to give our decision. Note that b_t is equivalent in the amount of information it carries with the information state I^t. In Equation 2.25, it is required to take the expectation of the states in the planning horizon. If at any time step, the information we have about the state s_t is its distribution b_t, then the result of the expectation would become a function of b_t. In mathematical terms, for a fixed action a, $E_{s_t}\left(r(s_t, a)|I^t\right) = b_t \mathbf{r}(a)$. Recall that $b_t = f(b_{t-1}, o_t, a_{t-1})$, where $f(.)$ is the function described in Equation 2.24. Then $b_t = \tau(b_0, I^t)$.

Assume that the value of b_{t-1} is known. Note that, if we have decided on an action for every possible value of the belief state, then we know which action

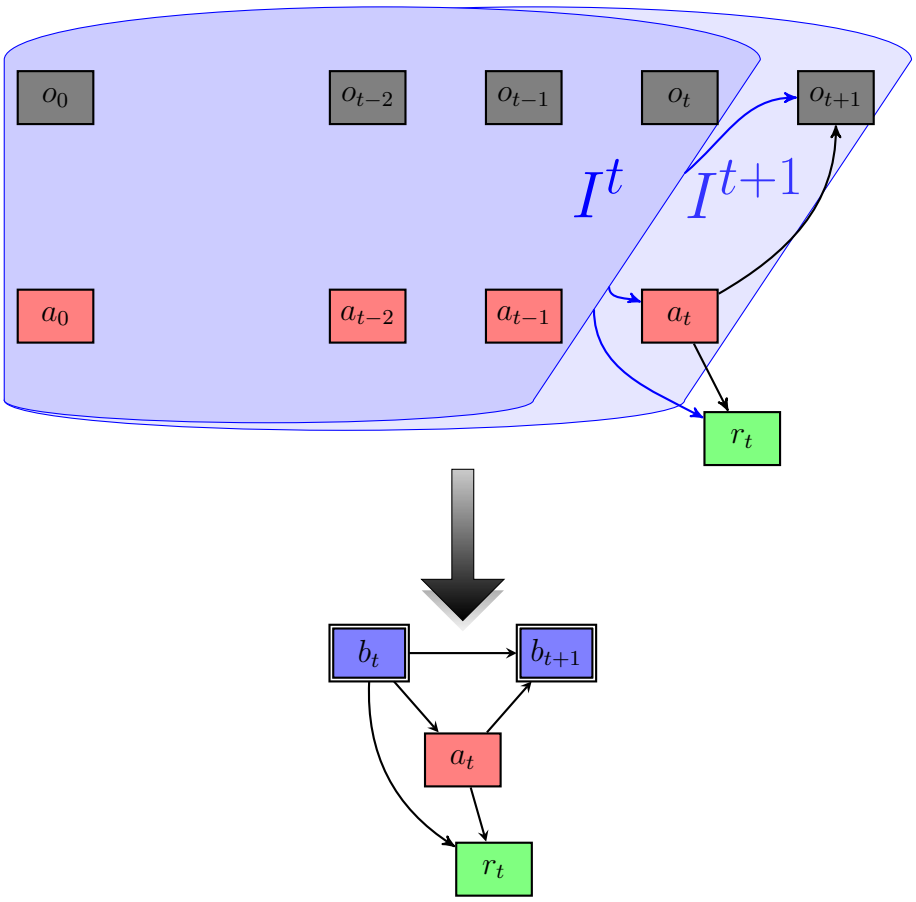


Figure 2.3: The first figure uses the POMDP framework for expressing the evolution of the complete information state at each timestep. We will call this figure as the Information State Influence Diagram. This complete information state at each timestep can be summarized as the belief state, which demonstrates the Markovian property. Thus, the second figure is called as the Belief State MDP Influence Diagram

to take for the particular value of b_{t-1}. After observing o_t, we can update the belief state to b_t. This shows us that, if we define our policy as $\mu_t(b) : \mathcal{B} \to \mathcal{A}$, the Markovian property will be preserved. Then, for a fixed planning horizon N and a given plan $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\} : \mathcal{B} \times \mathcal{B} \ldots \times \mathcal{B} \to \mathcal{A} \times \mathcal{A} \ldots \times \mathcal{A}$, the recursive formula in Equation 2.24 is calculable if the observation sequence $O^t := (o_0, \ldots, o_t)$ is known. Then we can write $b_t = f^\pi(b_0, O^t)$. Then, for a given observation sequence $O^t := (o_0, \ldots, o_t)$, the expected value of the immediate reward at time t for plan π would be:

$$E\left(r(s_t, a_t)\middle|b_0, O^t, \pi\right) = b_t \mathbf{r}(\mu_t(b_t)) \tag{2.26}$$

$$= f^\pi(b_0, O^t)\mathbf{r}(\mu_t(f^\pi(b_0, O^t))) \tag{2.27}$$

$$= g^\pi(b_0, O^t) \tag{2.28}$$

Therefore, for a fixed plan π, the immediate reward r_t becomes a function of observation sequence O^t and prior belief b_0. At time $t = 0$, as the observations are not known *in advance*, we need to take an expectation over O^t to calculate the expected value of r_t. Moreover, $O^{t+1:N-1} := (o_{t+1}, \ldots, o_{N-1})$ does not play any role on this calculation.

2.3.2 Dynamic Programming

Here we will define a iterative method for calculating the value function over the belief space $\mathcal{B}$. For any time t, if b_t is known, the values of O^t become irrelevant for the calculation of the expected value of r_k where $k > t$. Therefore define $V_t^\pi(b)$ for some fixed plan $\pi = \{\mu_0, \mu_1, \ldots, \mu_{N-1}\}$ as:

$$V_t^\pi(b) = E_{O^{t+1:N-1}}\left(r^\pi(s_N) + \sum_{k=t}^{N-1} r^\pi(s_k, \mu_k(b_k))\middle| b_t = b\right) \tag{2.29}$$

Note that

$$V_N^\pi(b) = b\mathbf{r}(\mu_N(b)) \tag{2.30}$$

Moreover,

$$V_t^{\pi}(b) = b\mathbf{r}(\mu_t(b)) + E_{o_{t+1}}\left(V_{t+1}^{\pi}(b_{t+1})\middle|b_t = b\right) \tag{2.31}$$

$$= b\mathbf{r}(\mu_t(b)) + E_{o_{t+1}}\left(V_{t+1}^{\pi}(f(b_t, o_{t+1}, a_t))\middle|b_t = b\right) \tag{2.32}$$

$$= b\mathbf{r}(\mu_t(b)) + E_{o_{t+1}}\left(V_{t+1}^{\pi}(f(b, o_{t+1}, \mu_t(b)))\right) \tag{2.33}$$

$$= b\mathbf{r}(\mu_t(b)) + \sum_{o\in\Theta} p(o|b, \mu_t(b))V_{t+1}^{\pi}(f(b, o, \mu_t(b))) \tag{2.34}$$

where $p(o|b,a)$ is the probability of observing o for the belief state b when action a is executed. Similar to Equation 2.24, we can write $p(o|b,\mu_t(b)) = b\mathbf{P}(\mu_t(b))\mathbf{D}(\mu_t(b),o)1$ as $\mu_t(b)$ and o are fixed values. As $\mu_t(b) \in \mathcal{A}$, there are only a finite number of possibilities for the policy selection for a given belief state b. Then switching to the optimal plan π^*,

$$V_t^*(b) = \max_{a\in\mathcal{A}}\left(b\mathbf{r}(a) + \sum_{o\in\Theta} p(o|b,a)V_{t+1}^*(f(b,o,a))\right) \tag{2.35}$$

where $\mathbf{r}(a)$ is the immediate reward function given in the vector form, V_t^* is the value function to be backed up at each time step. $f(b,o,a)$ is the updated belief state after action a is executed and observation o is experienced. As a reminder, the calculation of $f(b,o,a)$ is given in Equation 2.24. For notational convention, $b_o^a := f(b,o,a)$. Note that, as $V_{t+1}^*(.)$ is the optimal value function from time $t+1$ to N for all $b \in \mathcal{B}$, determining the optimal policy $\mu_t^*(.)$ becomes a separate problem then calculating $V_{t+1}^*(.)$.

2.4 Properties of the Value Function in POMDPs

Sondik and Smallwood [20] showed that the optimal finite horizon value function is piecewise linear and convex (PWLC) for any planning horizon N. PWLC property is useful because it allows the value function to be represented using finite resources. Assume that for some vector set Γ_{t+1}, the value function at time $t+1$ can be written as

$$V_{t+1}^*(b) = \max_{\boldsymbol{\gamma}\in\Gamma_{t+1}} b\cdot\boldsymbol{\gamma} \tag{2.36}$$

In this section, we want to state some properties of Equation 2.36 as these would be useful for defining $V_t^*(b)$. Defining $\boldsymbol{\gamma} = [\boldsymbol{\gamma}(1)\ldots\boldsymbol{\gamma}(D)]^T$, we will arrive

at $b.\boldsymbol{\gamma} := \sum_{l=1}^{D} b(l)\boldsymbol{\gamma}(l) = b\boldsymbol{\gamma}$. Therefore, the value function V_{t+1}^* in Equation 2.36 can be represented by a number of vectors represented by Γ_{t+1}.

One candidate for $V_{t+1}^*(b)$ when $D = 2$ is shown in Figure 2.4. As $b\mathbf{e} = 1$, the belief set $\mathcal{B}$ can be represented by a line. Each linear segment corresponds to a hyperplane over some closed subset of the belief set and can be represented by an D-vector of coefficients, which is shown as $\boldsymbol{\gamma}$ in Equation 2.36. While our aim here is to define the general properties of the value function for any given time step, the time index will be dropped and the set Γ_{t+1} will be denoted by $\bar{\Gamma} = \{\boldsymbol{\gamma}_i\}_{i=1}^{N}$, where $\boldsymbol{\gamma}_i \in \mathbb{R}_+^D$, $D \ll N$.

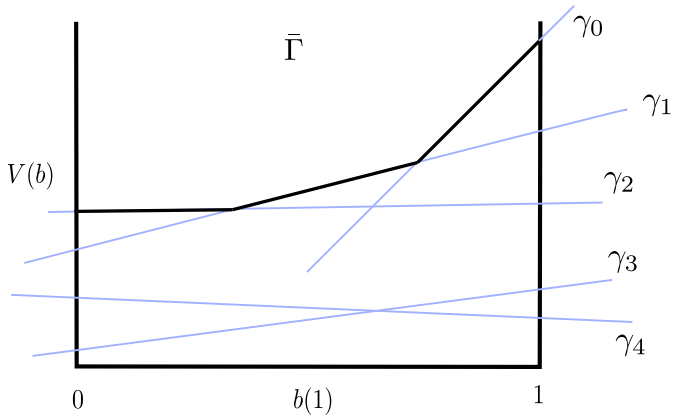


Figure 2.4: An example value function for a POMDP with two states

Each vector in Figure 2.4 corresponds to a policy tree. It is possible that a policy tree might represent the optimal strategy at some point in the belief space and contribute to the computation of the optimal value function. However, if a policy tree, or the vector representing it, is not optimal for any belief state, there is no need to keep that policy in memory. In mathematical terms,

Definition 3. *The support set of $\boldsymbol{\gamma}_i \in \bar{\Gamma}$ is defined as;*

$$R(\boldsymbol{\gamma}_i, \bar{\Gamma}) = \{b \in \mathcal{B} : b\boldsymbol{\gamma}_i > b\boldsymbol{\gamma}_j, \quad \forall \boldsymbol{\gamma}_j \in \bar{\Gamma} - \{\boldsymbol{\gamma}_i\}\} \tag{2.37}$$

The following property follows from the convexity of the value function.

Lemma 2.4.1. *The support set of any vector is a convex set.*

Proof. Suppose $b_1, b_2 \in R(\boldsymbol{\gamma}_i, \bar{\Gamma})$ and $\boldsymbol{\gamma}_j \in \bar{\Gamma}, i \neq j$. Now pick any belief state $b = \lambda b_1 + (1-\lambda)b_2, \quad 0 \leq \lambda \leq 1$.

$$\begin{aligned} b\boldsymbol{\gamma}_i &= (\lambda b_1 + (1-\lambda)b_2)\boldsymbol{\gamma}_i \\ &= \lambda b_1 \boldsymbol{\gamma}_i + (1-\lambda)b_2\boldsymbol{\gamma}_i \\ &> \lambda b_1 \boldsymbol{\gamma}_j + (1-\lambda)b_2\boldsymbol{\gamma}_j \\ &= b\boldsymbol{\gamma}_j \end{aligned}$$

■

Definition 4. *If $R(\boldsymbol{\gamma}_i, \bar{\Gamma}) = \emptyset$, then vector $\boldsymbol{\gamma}_i$ is dominated by the set of vectors $\bar{\Gamma}$.*

Therefore, for any $\boldsymbol{\gamma}_i \in \bar{\Gamma}$, if $\boldsymbol{\gamma}_i$ is dominated, then it can be deleted. A useless policy tree is equivalent to a dominated vector. Similarly if $R(\boldsymbol{\gamma}_i, \bar{\Gamma}) \neq \emptyset$, we will call it a non-dominated vector.

Definition 5. *Any point $b \in R(\boldsymbol{\gamma}_i, \bar{\Gamma})$ is called a witness point for vector $\boldsymbol{\gamma}_i$ relative to the set $\bar{\Gamma}$.*

Definition 6. *The witnessed vector for a belief state relative to the set $\bar{\Gamma}$ is defined by*

$$w(b, \bar{\Gamma}) := \arg\max_{\boldsymbol{\gamma}_i \in \bar{\Gamma}} b\boldsymbol{\gamma}_i$$

There can be more than one witness vectors in some belief state points, that is $w(b, \bar{\Gamma})$ is not a one-to-one function.

Therefore $b \in R(\boldsymbol{\gamma}_i, \bar{\Gamma}) \iff \boldsymbol{\gamma}_i \in w(b, \bar{\Gamma})$.

Definition 7. *A set $\bar{\Gamma}$ is called dirty if $\exists \boldsymbol{\gamma}_i \in \bar{\Gamma}$ such that $R(\boldsymbol{\gamma}_i, \bar{\Gamma}) = \emptyset$*

Definition 8. *For a given dirty set $\bar{\Gamma}$, the clean set, Γ, is defined as follows:*

$$\boldsymbol{\gamma}_i \in \Gamma \iff R(\boldsymbol{\gamma}_i, \bar{\Gamma}) \neq \emptyset \tag{2.38}$$

Therefore, $R(\boldsymbol{\gamma}_i, \bar{\Gamma}) = R(\boldsymbol{\gamma}_i, \Gamma)$. We will call this operation pruning and define it as follows;

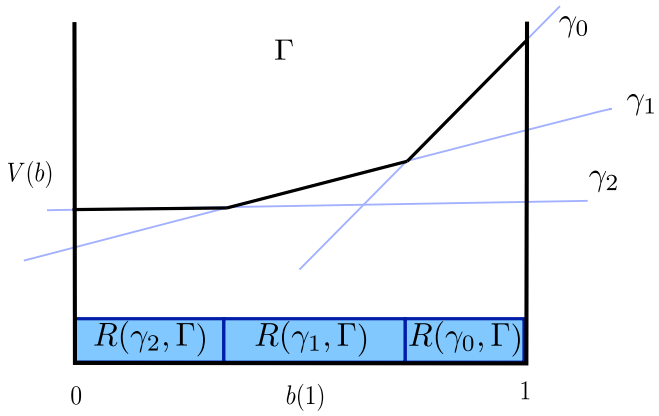


Figure 2.5: Belief State Partition

Definition 9. *The operator* $\mathbb{PR}(.)$ *finds the clean set for any given dirty set* $\bar{\Gamma}$. *Therefore,*

$$\Gamma = \mathbb{PR}(\bar{\Gamma})$$

Notice that the exact value function in Figure 2.4 can be represented by using the clean set of vectors as shown in Figure 2.5. This thesis is mainly about the pruning operator defined by $\mathbb{PR}(.)$. There are two legitimate questions that can be raised. The first question is; how *fast* is the pruning operator? For any given dirty set $\bar{\Gamma}$, how fast do we achieve the clean set Γ. This question will be answered in Chapter 3. The second question is; how *efficient* is the pruning operator? Note that, we haven't described so far the dirty set $\bar{\Gamma}$. We have only noted that there will be an increase while passing from the set Γ_{t+1} to Γ_t and then this set Γ_t should be pruned to a minimal set. If it is possible to create a smaller set Γ_t in the first hand, then the pruning operation will obviously be faster. This question will be answered in Chapter 4.

We can make an immediate observation for the pruning operation introduced in Definition 9 at this point. Note that in Figure 2.4, vector $\boldsymbol{\gamma}_4$ is never able to determine the value function, because for every value of $b \in \mathcal{B}$, $\boldsymbol{\gamma}_1$ would have a higher value. This basic mechanism for vector pruning is called as pointwise domination and can be formally shown as follows;

Lemma 2.4.2. *Let* $\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_j \in \bar{\Gamma}$. *If* $\boldsymbol{\gamma}_i(l) > \boldsymbol{\gamma}_j(l)$, $1 \leq l \leq D$, *then vector* $\boldsymbol{\gamma}_j$ *is dominated by* $\boldsymbol{\gamma}_i$.

Proof. For any $b \in \mathcal{B}$, we can write $b\boldsymbol{\gamma}_i > b\boldsymbol{\gamma}_j$ as $b \geq \mathbf{0}^T$. ■

However, it is not possible to prune all dirty vectors and arrive at the clean set shown in Figure 2.5 by only using Lemma 2.4.2. The pruning operation will be discussed in detail in Chapter 3. But before going into the details of the pruning operation, we want to discuss how the vectors are created in each time step of the planning horizon in the following section.

2.5 Exact Value Iteration

Recall that our aim is to calculate the optimal value function $J^*(b), \forall b \in \mathcal{B}$ for a fixed planning horizon N. For this, we start from $V_N^*(b)$ and try to calculate $V_0^*(b) = J^*(b)$. As seen from the previous section, this means to compute V_t^* from V_{t+1}^*. We will now show that this is equivalent to compute the set Γ_t from the set Γ_{t+1}.

Using Equations 2.35 and 2.36 and defining $\mathbf{R}(a,o) := \mathbf{P}(a)\mathbf{D}(a,o)$,

$$V_t^*(b) = \max_{a \in \mathcal{A}} \left(b\mathbf{r}(a) + \sum_{o \in \Theta} p(o|b,a) \max_{\boldsymbol{\gamma} \in \Gamma_{t+1}} b_o^a \boldsymbol{\gamma} \right) \tag{2.39}$$

$$V_t^*(b) = \max_{a \in \mathcal{A}} \left(b\mathbf{r}(a) + \sum_{o \in \Theta} p(o|b,a) b_o^a w(b_o^a, \Gamma_{t+1}) \right) \tag{2.40}$$

$$V_t^*(b) = \max_{a \in \mathcal{A}} \left(b\mathbf{r}(a) + \sum_{o \in \Theta} b\mathbf{R}(a,o)\mathbf{e} \frac{b\mathbf{R}(a,o)}{b\mathbf{R}(a,o)\mathbf{e}} w(b_o^a, \Gamma_{t+1}) \right) \tag{2.41}$$

$$V_t^*(b) = \max_{a \in \mathcal{A}} \left(b\mathbf{r}(a) + \sum_{o \in \Theta} b\mathbf{R}(a,o) w(b_o^a, \Gamma_{t+1}) \right) \tag{2.42}$$

We want to show that it is possible to find a vector set Γ_t such that,

$$V_t^*(b) = \max_{\boldsymbol{\gamma} \in \Gamma_t} b\boldsymbol{\gamma} \tag{2.43}$$

Then it easily follows that,

$$J^*(b) = V_0^*(b) = \max_{\boldsymbol{\gamma} \in \Gamma_0} b \cdot \boldsymbol{\gamma} \tag{2.44}$$

Note that we have not discussed yet how to find the set Γ_t described in Equation 2.43. Before going into the mathematical details of arriving from Equation 2.42

to Equation 2.43, we want to discuss the procedure verbally. Note that we can first calculate 2.42 for each $a \in \mathcal{A}$ and then take the outer maximization. Then for a fixed value of a, since the first summand is fixed, if the second summand is PWLC then the whole summation would be PWLC. Finally for the second summand, the summation of a set of PWLC functions is a also PWLC function.

In mathematical terms, optimal finite horizon POMDP value function given in Equation 2.35 can be written in a series of related value functions in terms of vector operations;

$$V_t^*(b) = \max_{a \in \mathcal{A}} V_t^{*,a}(b) \tag{2.45}$$

$$V_t^{*,a}(b) = \sum_{o \in \Theta} V_t^{*,a,o}(b) \tag{2.46}$$

$$V_t^{*,a,o}(b) = \frac{1}{|\Theta|} b\mathbf{r}(a) + p(o|b,a) V_{t+1}^*(b_o^a) \tag{2.47}$$

Equations 2.45, 2.46, 2.47 show a way to decompose Equation 2.35 into smaller parts while calculating one step of the dynamic programming algorithm. Using Equations 2.47 and 2.42 and Definition 6,

$$V_t^{*,a,o}(b) = \frac{1}{|\Theta|} b\mathbf{r}(a) + b\mathbf{R}(a,o) w(b_o^a, \Gamma_{t+1}) \tag{2.48}$$

In Equation 2.48, there is a nontrivial maximization $w(b_o^a, \Gamma_{t+1})$ which should be calculated for the whole belief set $\mathcal{B}$ for any fixed values of a and o. As Sondik and Smallwood note [20], $w(b_o^a, \Gamma_{t+1})$ is a finitely valued function of b. As $V_{t+1}^*(.)$ is a convex function and b_o^a is a continuous function of b, $w(b_o^a, \Gamma_{t+1})$ partitions the belief set into a finite number of regions such that $w(b_o^a, \Gamma_{t+1})$ is single valued over each region.

Note that the calculation of Equation 2.48 should be repeated for $|\mathcal{A}| \times |\Theta|$ times. Once the vector that gives the maximum value for each belief state $b \in \mathcal{B}$ is found, Equations 2.45, 2.46 should also be performed for the whole belief set $\mathcal{B}$. It is also possible to write Equation 2.48 in vector form as follows,

$$\boldsymbol{\gamma}_t^{a,o}(b) = \frac{1}{|\Theta|} \mathbf{r}(a) + \mathbf{R}(a,o) w(b_o^a, \Gamma_{t+1}) \tag{2.49}$$

Note that $\boldsymbol{\gamma}_t^{a,o}$ here is a function of b, due to the term $w(b_o^a, \Gamma_{t+1})$. Assume that $\boldsymbol{\gamma}_i = w(b_o^a, \Gamma_{t+1})$. As we know that $R(\boldsymbol{\gamma}_i, \Gamma_{t+1})$ is a connected subset of $\mathcal{B}$, we can

see that this function has a constant value as long as $b_o^a \in R(\boldsymbol{\gamma}_i, \Gamma_{t+1})$. Then,

$$\boldsymbol{\gamma}_t^a(b) = \sum_{o \in \Theta} \boldsymbol{\gamma}_t^{a,o}(b) \tag{2.50}$$

Finally,

$$d_t(b) = \arg\max_{a \in \mathcal{A}} b\boldsymbol{\gamma}_t^a(b) \tag{2.51}$$

$$\boldsymbol{\gamma}_t(b) = \boldsymbol{\gamma}_t^{d_t(b)}(b) \tag{2.52}$$

2.6 The Computational Complexity of Exact Value Iteration

To have an idea about the complexity of the problem, we will first figure out an upper bound for the number of calculations needed. Assume that we seek to find the expected reward for a known belief state b at time t. From Equation 2.49, we can see that it is necessary to calculate b_o^a for any selection of o and a. Assuming that we have a stationary system model (i.e., fixed $\mathbf{P}(a)$ and $\mathbf{D}(a, o)$ matrices), we can prepare the matrices $\mathbf{R}(a, o) = \mathbf{P}(a)\mathbf{D}(a, o)$ for each selection of o and a. The multiplication $b\mathbf{R}(a, o)$ amounts to D^2 multiplications and $D \times (D-1)$ additions. Therefore, the complexity of calculating each b_o^a is $O(|\mathcal{A}| \times |\Theta| \times D^2)$. If Γ_{t+1} is known, then the calculation of $w(b_o^a, \Gamma_{t+1})$ is possible by $|\Gamma_{t+1}| \times D$ multiplications and $|\Gamma_{t+1}| \times (D-1)$ additions. As the maximum vector should be found for each b_o^a, the total complexity becomes $O(|\Gamma_{t+1}| \times D \times |\mathcal{A}| \times |\Theta| \times D^2)$

In a recursive manner, for a fixed planning horizon N and a prior belief $b = b_0$, the computational complexity of calculating $J^*(b_0)$ would be $O(|\mathcal{A}|^N \times |\Theta|^N \times D^{2N})$. The complexity due to calculating $w(b_o^a, \Gamma_{t+1})$ is discarded in this case, as it is not necessary to calculate $|\Gamma_t|,\ 0 \leq t \leq N$, to find the value of $J^*(b_0)$. However, this number of operations are necessary for each belief point selected. Now assume that, we have calculated the set Γ_0. Then, the expected accumulated reward could be calculated by the vector multiplications described in Equation 2.44 which amounts to a complexity of $O(D \times |\Gamma_0|)$. As the number of belief states are uncountably infinite, after the number of belief states exceeds a certain number, it will become cost effective to calculate the set Γ_0 then calculating point based value iterations for each belief state. This is surely dependent on

the problem definition. For instance, if the system model is not stationary (i.e., the transition probability $p_t(s'|s,a)$ or the observation probability $p_t(o|s,a)$ is changing with time), the set of vectors in $\Gamma_{0:N}$ cannot be used for the value function calculation in the future time steps.

Therefore it is important to calculate Γ_0 for a given planning horizon N. In an iterative manner, this problem is equivalent to calculating Γ_t when Γ_{t+1} is given. Equation 2.36 shows that the set Γ_t will be a finite set of vectors, but the series of equations does not immediately show how to find these vectors. The idea proposed by Monahan in [21] gives an upper bound of the complexity of exact value iteration algorithms. The algorithm creates a large number of candidate vectors and then get rid of the useless ones. Due to the expensive operation which calculates the vectors, it is called the Enumeration Algorithm.

For this aim, we will look at Equation 2.49 in a different way. Note that there are only a finite number of possible values for $w(b_o^a, \Gamma_{t+1})$ and we select the vector that gives the highest value. Therefore if we take all the vectors in set Γ_{t+1} and multiply them by $\mathbf{R}(a,o)$ we arrive to a new set of vectors. For this new set of vectors, the maximum value for b is the same as $w(b_o^a, \Gamma_{t+1})$.

In mathematical terms, recall that in Equation 2.49, $w(b_o^a, \Gamma_{t+1}) \in \Gamma_{t+1}$. Assume that we have created the set

$$\bar{\Gamma}_t^{a,o} = \left\{ \frac{1}{|\Theta|}\mathbf{r}(a) + \mathbf{R}(a,o)\gamma_{t+1} \mid \gamma_{t+1} \in \Gamma_{t+1} \right\} \tag{2.53}$$

Then for any $b \in \mathcal{B}$, $\boldsymbol{\gamma}_t^{a,o}(b) \in \bar{\Gamma}_t^{a,o}$. Moreover, $\boldsymbol{\gamma}_t^{a,o}(b) = w(b, \bar{\Gamma}_t^{a,o})$. Using Definition 9,

$$\Gamma_t^{a,o} = \mathbb{PR}\left(\bar{\Gamma}_t^{a,o}\right) \tag{2.54}$$

Then, $\boldsymbol{\gamma}_t^{a,o}(b) = w(b, \Gamma_t^{a,o})$.

Now we know that for each a and o, the vectors described in Equation 2.53 forms $V_t^{*,a,o}(.)$ in Equation 2.47 which is a convex function. For Equation 2.46, we need to take sum of PWLC functions which is a PWLC function. We know that, there are a finite number of vectors in each set $\Gamma_t^{a,o}$, $o \in \Theta$. Therefore, if we create all possible vectors from the combination of these sets, the vectors described in Equation 2.50 would be in these vectors. Such an approach would

avoid dependency to particular belief states. For the mathematical description, we will first define the cross-sum operator.

Definition 10. *For set of vectors $\mathcal{U}$ and $\mathcal{V}$, the cross sum operator $\oplus$ is defined by*

$$\mathcal{U} \oplus \mathcal{V} = \{u + v | u \in \mathcal{U}, v \in \mathcal{V}\}$$

When there are more than two sets of vectors $\{\mathcal{U}_i\}_{i=1}^{M}$,

$$\bigoplus_i \mathcal{U}_i = \{\ldots\{\mathcal{U}_1 \oplus \mathcal{U}_2\}\ldots\mathcal{U}_M\}$$

Using Definitions 9 and 10 we can write,

$$\bar{\Gamma}_t^a = \bigoplus_{o \in \Theta} \Gamma_t^{a,o} \tag{2.55}$$

$$\Gamma_t^a = \mathbb{PR}\left(\bar{\Gamma}_t^a\right) \tag{2.56}$$

A similar approach is also possible for the vectors in Equation 2.52. We take an union of all vectors in sets Γ_t^a and then find the value function in Equation 2.45 by taking the ones that are not dominated. That is,

$$\bar{\Gamma}_t = \bigcup_{a \in \mathcal{A}} \Gamma_t^a \tag{2.57}$$

$$\Gamma_t = \mathbb{PR}\left(\bar{\Gamma}_t\right) \tag{2.58}$$

where $\bigcup$ is the union operator. It is easy to show that $\boldsymbol{\gamma}_t^a(b) \in \Gamma_t^a$ and $\boldsymbol{\gamma}_t(b) \in \Gamma_t$ for any $b \in \mathcal{B}$.

As noted in [51], Monahan's Enumeration Algorithm provides us an upper bound for the exact value iteration algorithms. In Equation 2.53, each vector should be multiplied by $\mathbf{R}(a, o)$ which takes D^2 multiplications and $D \times (D-1)$ additions. Moreover, this operation should be repeated for each a and o. This amounts to $O(|\Gamma_{t+1}| \times |\mathcal{A}| \times |\Theta| \times D^2)$.

After vectors $\Gamma_t^{a,o}$ are found, we need to find Γ_t^a in Equation 2.55. Note that summation of two vectors is D additions. From each set there are $|\Gamma_{t+1}|$ different possibilities to add while creating a vector, therefore the total operations is $O(|\mathcal{A}| \times D \times |\Gamma_{t+1}|^{|\Theta|})$ new vectors for each action. Hence, the overall complexity

of calculating all vectors for a single step of enumeration algorithms is $O(|\mathcal{A}| \times D \times |\Gamma_{t+1}|^{|\Theta|} + |\Gamma_{t+1}| \times |\mathcal{A}| \times |\Theta| \times D^2)$.

Comparing this result to the complexity results for finding the value of $J^*(b_0)$ for a fixed prior belief b_0 might be useful. Note that the enumeration algorithm is very costly as $O(|\Gamma_0|^{|\Theta|}) \approx O(|\Gamma_N|^{N \times |\Theta|}) \approx O(|\mathcal{A}|^{N \times |\Theta|})$. While the number of operations were also exponentially increasing in the former case, here the rate of exponential increase is $|\Theta|$ times faster. Therefore it is fundamentally important both to limit the number of vectors created while passing from Γ_{t+1} to $\bar{\Gamma}_t$ and also do the pruning of unnecessary vectors in $\bar{\Gamma}_t$. The following chapters will consider these two problems independently.

CHAPTER 3

AN ALGEBRAIC AND GEOMETRIC FRAMEWORK FOR VECTOR PRUNING IN POMDPS

3.1 Introduction

In Chapter 2, an upper bound for the computational complexity of exact value iteration algorithm was discussed. It was shown that, if all possible vectors were created in each step of the exact value iterations, the number of vectors increases exponentially with $|\Theta|$ and linearly with $|\mathcal{A}|$. Yet, not all these vectors are useful when determining the optimal value function V_n^*. Only those vectors that are maximal at some belief state are really necessary. The process of finding these vectors is called *pruning*.

Having a fast pruning procedure which can take a set of vectors $\bar{\Gamma}$ and reduce it to a set of non-dominated vectors Γ, is the main concern for exact value iteration. Although the vector formation procedure in POMDPs is of a special kind (transforming the vector by multiplying with matrix $\mathbf{R}(a, o)$ and then the cross-sum addition of different vectors), we will start analyzing the vector pruning problem assuming that we have a random set of vectors. Our analysis will start by the comparison of two pruning algorithms from the literature; the Lark's algorithm [24] and Skyline algorithm [51]. The linear programming structure of both of the algorithms will be introduced and the degrees of freedom on both the selection of the constraints and the objective function of each linear program will be discussed.

This chapter will then continue with a geometric framework of the pruning

procedure. It is possible to represent the value function over the belief set and in the vector space. In the belief set representation, each vector describes a hyperplane equation. If the vector is non-dominated, this hyperplane equation is used to define the optimal value function in some convex region of the belief state set. In the vector space representation, the optimal value function forms a convex polyhedron [52] where each hyperplane equation belonging to a non-dominated vector is represented by a vertex of an upper convex polyhedron [53]. There is a field of research in the convex analysis and computational geometry literature regarding the properties of convex polyhedrons [53–55]. If the set of all vertices are traversed, we get an edge graph of the polyhedron which can be used to find the dominated vectors [56,57]. The vectors that are not in this edge graph will be dominated. However, this operation is tedious for the pruning procedure where the aim is to find only the set of non-dominated vectors.

We will use the dual representation of the problem for a different purpose. We will demonstrate that the linear programming iterations refer to the selection of different convex regions in the vector space representation of the value function. This convex region can be used to determine if any vectors from the dirty set are dominated. Instead of writing all the constraints, the dirty vectors that are most likely to be dominated in this simplex iterations are written to the simplex tableau. This will allow us to decompose the problem into small sized LPs. We name this as the algebraic framework, which is the construction and utilization of linear programs. The relation between the algebraic and geometric frameworks have been exploited in this study.

Finally, we offer a novel pruning algorithm, called FastCone, based on the theoretical framework stated in this thesis. For a given set of clean vectors the algorithm hastily searches for the convex region that a dirty vector is in and searches for a clean vector if only the set of clean vectors is not sufficient to prune this dirty vector. To show the algorithm performance, FastCone algorithm is compared to the existing algorithms and their revised versions both analytically and experimentally.

This chapter is organized as follows. Section 3.3 introduces the algebraic and

geometric approaches for the vector pruning problem and contains the mathematical insight that led to the FastCone and Section 3.4 gives the pseudocode for the algorithm. Section 3.5 gives the test results on both the artificial and benchmark problems. Section 3.6 concludes the chapter.

3.1.1 Conventions

Before introducing different pruning algorithms, we want to define some conventions that will be used throughout the chapter. We will denote $\bar{\Gamma}$ as the dirty set and Γ as the clean set. Assume that $\bar{\Gamma} = \{\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_N\}$. We will denote the vectors in the set $\bar{\Gamma}$ by their index set $F_0 = \{1, \ldots, N\}$. As we give our decision about the vectors in the dirty set $\bar{\Gamma}$, the size of F_0 decreases. For notational convention, we will define F, which represents the index of current dirty set of vectors, respectively. At initialization, the dirty set contains all the vectors, $F = F_0$. All algorithms continue until $F = \emptyset$. In a similar manner, we will define Q_∞ and Q which describe the final and current clean set of vectors, respectively. The indexes of the vectors in Γ are one-to-one correspondent with the set Q_∞.If any index set A is used as a subscript of a vector set, i.e., $\bar{\Gamma}_A$, this shows that from the set $\bar{\Gamma}$, the vectors with the indices in A are selected. With this definition, we can write $\Gamma = \bar{\Gamma}_{Q_\infty}$. At the beginning of the algorithm, $Q = \emptyset$ and when the algorithm is terminated, $Q = Q_\infty$.

3.2 Known Pruning Algorithms

We have selected two pruning algorithms from the literature that would allow us to introduce the necessary concepts for defining the vector pruning problem. The mathematical formulations introduced for the algorithms will then be used for introduction of a novel algorithm.

3.2.1 Lark's Algorithm

When an arbitrary set of vectors is given, Lark's algorithm starts with $F = F_0$ and $Q = \emptyset$. The algorithm picks a vector $\boldsymbol{\gamma}_i \in \bar{\Gamma}(F)$ and tries to find a belief point b that satisfies $b\boldsymbol{\gamma}_i > b\boldsymbol{\gamma}_j$, $\forall \boldsymbol{\gamma}_j \in \bar{\Gamma}_Q$. Such a belief point is found by the following LP:

$$\begin{aligned} &\min \delta \\ &b(\boldsymbol{\gamma}_i - \boldsymbol{\gamma}_j) + \delta > 0, \quad \forall j \in Q \\ &\sum_{l=1}^{D} b(l) = 1 \\ &b(l) \geq 0, \quad 1 \leq l \leq D \end{aligned} \tag{3.1}$$

The optimal solution occurs at the belief state b_0 and the value of the objective function is δ_0. If δ_0 is less than 0, it means that there is a vector in set $\bar{\Gamma}$ that gives a higher value for the belief state b_0 where the optimal solution occurs. The vector index $k' = \arg\max_{\boldsymbol{\gamma}_k \in \bar{\Gamma}} b_0 \boldsymbol{\gamma}_k$ is added to the clean set Q and deleted from F. If δ_0 is greater than or equal to zero, the vector $\boldsymbol{\gamma}_i$ is dominated by the vectors in the clean set $\bar{\Gamma}_Q$ and therefore i is deleted from F. The procedure continues until there are no vectors left in F. The number of constraints in the LP is $|Q|$, therefore as $|Q|$ gets larger, the LP becomes harder to solve.

Algorithm 1 is the Lark's algorithm. The main routine is LRK, where we get an arbitrary set of vectors, $\bar{\Gamma}$, and initialize an empty clean set Γ. After a new vector, $\boldsymbol{\gamma}$ is selected from the dirty set, we start the linear program discussed by the FNDBLF procedure. FNDBLF procedure tries to find a witness point for the given vector $\boldsymbol{\gamma}$ with respect to the set $\bar{\Gamma}_Q$. At the end of the optimization is δ is negative, b is a witness point of the vector $\boldsymbol{\gamma}$.

Algorithm 1 also explains two other routines; PNTDOM and BEST. These two routines are used in the same fashion as the original algorithm. PNTDOM is used to prune, if possible, some of the dominated vectors without using linear programming which is described in Lemma 2.4.2. BEST is used to select one of the dominating vectors if a belief state is given. The symbol $<_{lex}$ in the pseudo-code denotes lexicographic ordering [58]. Lexicographic ordering is given in Algorithm 2.

Algorithm 1 Lark's Algorithm

```
1: procedure LRK(Γ̄)
2:     Q ← ∅
3:     F ← F_0
4:     while F ≠ ∅ do
5:         γ ← any element in Γ̄
6:         i ← index of γ in Γ̄
7:         if PNTDOM(γ, Γ̄_Q) then
8:             F ← F \ {i}
9:         else
10:            (δ, b) ← FNDBLF(γ, Γ̄_Q)
11:            if δ > 0 then
12:                F ← F \ {i}
13:            else
14:                γ̂ ← BEST(b, Γ̄)
15:                k ← index of γ̂ in Γ̄
16:                F ← F \ {k}
17:                Q ← Q ∪ {k}
18:            end if
19:        end if
20:    end while
21:    return Γ̄_Q
22: end procedure
23: procedure PNTDOM(γ, Γ)
24:    for all γ̂ ∈ Γ do
25:        if γ(l) ≤ γ̂(l), 1 ≤ l ≤ D then
26:            return true
27:        end if
28:    end for
29:    return false
30: end procedure
31: procedure BEST(b, Γ̄)
32:    γ̂ ← ∅
33:    k = −∞
34:    for all γ ∈ Γ̄ do
35:        if k < bγ then
36:            γ̂ ← γ
37:        else
38:            if k = bγ & γ̂ <_lex γ then
39:                γ̂ ← γ
40:            end if
41:        end if
42:    end for
43:    return γ̂
44: end procedure
45: procedure FNDBLF(γ, Γ)
46:    solve the following linear program
       variables: δ, b
       min δ  subject to
       b(γ − γ̂) + δ > 0   ∀γ̂ ∈ Γ
       Σ b(l) = 1
       b ≥ 0
47:    return (δ, b)
48: end procedure
```

Algorithm 2 Lexicographic Ordering

```
1: procedure LEX(γ̂, γ)
2:     for all 1 ≤ l ≤ D do
3:         if γ̂(l) < γ(l) then
4:             return γ
5:         end if
6:         if γ(l) < γ̂(l) then
7:             return γ̂
8:         end if
9:     end for
10:    return γ̂
11: end procedure
```

3.2.2 Skyline Algorithm

An alternative to the Lark's algorithm is the Skyline algorithm proposed by Raphael and Shani [51]. Skyline algorithm traces the upper envelope formed by the set of vectors $\bar{\Gamma}$. All vectors visited during this traversal are non-dominated, hence should be added to the clean set Γ, while vectors that can never be visited are pruned.

The visualization is easier with a geometric description of the algorithm. In Figure 3.1, the partition of the belief state space $\mathcal{B}$ is shown for $D = 3$. As can be seen from the figure, all vectors have convex support sets as stated in Lemma 2.4.1. Any vertex on this graph can be represented by a set of equations. For instance, the belief state b marked on the graph is the solution for $b\boldsymbol{\gamma}_3 = b\boldsymbol{\gamma}_4 = b\boldsymbol{\gamma}_5$ and the simplex constraint $b\mathbf{e} = 1$. Now if we set one of the constraints free (for instance leaving $\boldsymbol{\gamma}_3$ would mean we are left with $b\boldsymbol{\gamma}_4 = b\boldsymbol{\gamma}_5$ and the simplex constraint $b\mathbf{e} = 1$), the set of equations will describe one of the lines emanating from point b and these lines would end at another vertex on the skyline. Repeating this strategy and keeping the visited points in a list, all possible vertices on the skyline can be visited. The vectors that are not visited during this traverse are the dominated ones.

In mathematical terms, this can be explained as follows. When an arbitrary set of vectors $\bar{\Gamma}$ is given, it is possible to write the following equations for any belief state $b \in \mathcal{B}$;

$$\begin{aligned} b\boldsymbol{\gamma}_i + x_i = b\boldsymbol{\gamma}_j + x_j \quad \forall i, j \in F_0 \\ x_i \geq 0, \quad \forall i \in F_0 \end{aligned} \tag{3.2}$$

where x_i, x_j are the slack variables. If we are at $b \in R(\boldsymbol{\gamma}_i, \bar{\Gamma})$, then we can satisfy the set of Equations 3.2 for $x_i = 0$ and conclude that vector $\boldsymbol{\gamma}_i$ is on the skyline. However, if $\boldsymbol{\gamma}_i$ is a dominated vector, it is not possible to satisfy the set of Equations 3.2 for $x_i = 0$. If all vertices in the unit simplex are traversed, the vectors for which $x_i \neq 0$ will be the dominated ones.

The operations done for the Skyline algorithm is equivalent to the enumeration of vertices of a convex polyhedron [56,57]. The difficulty with this approach is in

determining whether or not a vertex has been visited. Avis et al. [52] has offered using the criss-cross algorithm [59] which is used to guarantee that all vertices of the convex polyhedron are visited only once. The algorithm initializes a simplex tableau and uses the same tableau for the whole enumeration algorithm. Algorithm starts from a feasible solution and traverses a branch using the unique path described by the criss-cross algorithm. Once the end of that branch is reached, it traverses back until the first feasible solution is reached again. It is also possible to save some of the dictionaries. At the end, the algorithm produces a list that is free of duplicates even for degenerate inputs.

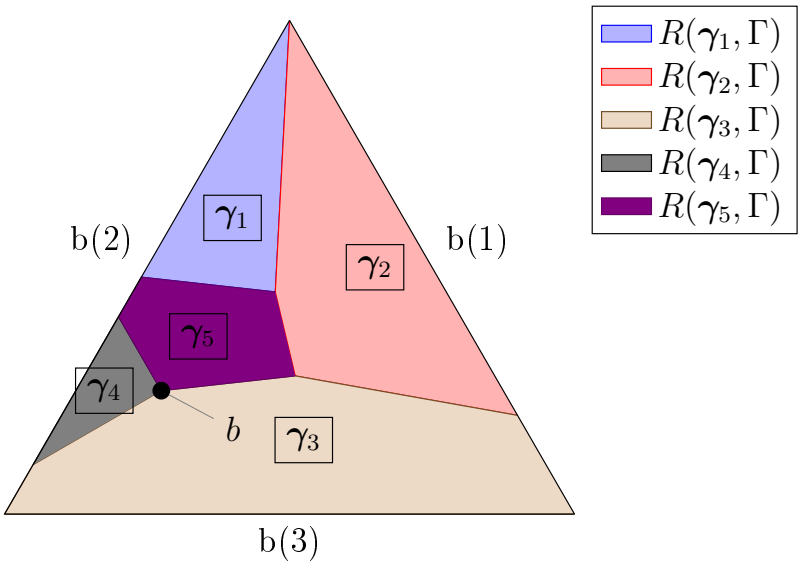


Figure 3.1: Belief State Representation for $D = 3$

Still, traversing all the vertices on the skyline can be costly for the pruning operation. For this, Raphael and Shani have offered an iterative version of the Skyline algorithm. Rather than visiting all possible vertices, this revised algorithm goes only in directions in which a predetermined vector γ_i comes closer to the skyline. The LP given in Equation 3.3 is called iterative Skyline algorithm [51].

$$
\begin{aligned}
&\min x_i \\
&b(\boldsymbol{\gamma}_i - \boldsymbol{\gamma}_j) + x_i - x_j = 0, \quad \forall j \in F_0 \setminus \{i\} \\
&\sum_{l=1}^{D} b(l) = 1 \\
&b(l) \geq 0, \quad 1 \leq l \leq D \\
&x_j \geq 0, \quad 1 \leq j \leq N
\end{aligned}
\tag{3.3}
$$

As we are always at a feasible solution, the value of x_i is always non-negative. For the smallest possible value of x_i, if the vector is still not on the skyline, vector index i is removed from F. If $x_i = 0$, vector index i is added to Q. After the decision about vector $\boldsymbol{\gamma}_i$ is given, the algorithm picks one of the other vectors, say $k \in F$ and changes the objective function to minimize x_k. The algorithm continues until $F = \emptyset$.

Note that, the constraints for the LP given in Equation 3.3 are written for the vector $\boldsymbol{\gamma}_i$. We can write the LP for $\boldsymbol{\gamma}_k$ instead of $\boldsymbol{\gamma}_i$ by applying linear row operations. After writing all the constraints for $\boldsymbol{\gamma}_k$, by only changing the objective function to minimize x_k, we can continue from the set of equations defining this point on the belief state.

Algorithm 3 is the iterative Skyline algorithm. The main procedure is defined by ISKY, where we get an arbitrary set of vectors $\bar{\Gamma}$. The dirty and clean set indices are initiated as $F = F_0$ and $Q = \emptyset$. LPINIT procedure writes the initial simplex tableau $\mathbf{P}$ defined by Equation 3.2. An initial feasible solution for this set of equations can be found when $x_j = 0$ where $j = \arg\max_{\boldsymbol{\gamma}_i \in \bar{\Gamma}} \boldsymbol{\gamma}_i(1)$ and $b(1) = 1$. While $b(l) = 0, l \in \{2, \ldots, D\}$ there are D variables equal to zero. Therefore this is a basic feasible point for the simplex matrix $\mathbf{P}$.

Note that through LPINIT procedure, an equation is defined for each vector in the set $\bar{\Gamma} - \{\boldsymbol{\gamma}_j\}$. This set of equations, with the simplex constraint $\sum_l b(l) = 1$, defines the simplex tableau. Therefore $\mathbf{P}$ is a $N \times (N+D)$ matrix. After LPINIT, the same simplex tableau is used until the end of the pruning procedure.

After the simplex tableau is initialized, the objective function is selected as

Algorithm 3 Iterative Skyline Algorithm

1: **procedure** ISKY($\bar{\Gamma}$)
2: $Q \leftarrow \emptyset$, $F \leftarrow F_0$
3: $j = \arg\max_{i \in F} \gamma_i(1)$
4: $F \leftarrow F \setminus \{j\}$, $Q \leftarrow Q \cup \{j\}$
5: $\mathbf{P} \leftarrow$ LPINIT($\bar{\Gamma}, j$)
6: **while** $F \neq \emptyset$ **do**
7: $i \leftarrow$ any element in F
8: $\mathbf{P} \leftarrow$ LPOBJSET($\mathbf{P}, i$)
9: **while** $i \in F$ **do**
10: $(\mathbf{P}, F, Q) \leftarrow$ LPITER($\mathbf{P}, F, Q, i$)
11: **end while**
12: **end while**
13: **return** $\bar{\Gamma}_Q$
14: **end procedure**
15: **procedure** LPINIT($\bar{\Gamma}, j$)
16: write the initial tableau $\mathbf{P}$
variables: $b, x_i,\ i \in F$
min 0
$b(\gamma_j - \gamma_i) + x_j - x_i = 0,\ \forall i \in F \setminus \{j\}$
$\sum_{l=1}^{D} b(l) = 1$
17: **return** $\mathbf{P}$
18: **end procedure**
19: **procedure** LPOBJSET($\mathbf{P}, i$)
20: set the objective function to $\min x_i$
21: **return** $\mathbf{P}$
22: **end procedure**
23: **procedure** LPITER($\mathbf{P}, F, Q, i$)
24: do one simplex iteration to $\mathbf{P}$
25: **for all** $j \in F$ **do**
26: **if** $x_j = 0$ **then**
27: $Q \leftarrow Q \cup \{j\}$
28: $F \leftarrow F \setminus \{j\}$
29: **end if**
30: **end for**
31: **if** x_i optimal **then**
32: **if** $x_i \neq 0$ **then**
33: delete the constraint row with x_i
34: **else**
35: $Q \leftarrow Q \cup \{i\}$
36: **end if**
37: $F \leftarrow F \setminus \{i\}$
38: **end if**
39: **return** $(\mathbf{P}, F, Q)$
40: **end procedure**

$\min x_i$, which is the slack variable of the hyperplane equation formed by $\boldsymbol{\gamma}_i$ by the function LPOBJSET. The objective function is important because it determines the direction of simplex iterations. LPITER is a simple simplex iteration that moves from one vertex to another. At every visited vertex, we check the non-dominated vectors and add their indices to Q if they were not added before. We also check if the optimal value is reached for the slack variable of the predetermined vector $\boldsymbol{\gamma}_i$. At its optimal value, the index of the vector $\boldsymbol{\gamma}_i$ is deleted from the set F. This index is added to the clean set Q depending on the value of the slack variable. The algorithm continues until there are no vector indices in the dirty set F.

3.2.3 Comparison of the Pruning Algorithms

Note that there is a great similarity between the LP given in Equation 3.1 and 3.3. However, there are two major differences. First one is the number of constraints in the LPs. While constructing the LP, Lark's algorithm compares vector $\boldsymbol{\gamma}_i$ to the vectors in the clean set Γ; whereas Skyline algorithm compares $\boldsymbol{\gamma}_i$ with all the vectors in the initial set $\bar{\Gamma}$. This is a disadvantage considering the time spent in the LP, but it also has a major advantage. The objective function of the LP in Equation 3.3 is to find whether if $\boldsymbol{\gamma}_i$ is on the skyline. But as LP progresses from one feasible solution to another, the simplex iteration reveals one of the non-dominated vectors. This is possible because LP considers all vectors in the set $\bar{\Gamma}$ and therefore, every basic feasible point of LP 3.3 is definitely on the skyline. Moreover, there is no need to write the LP from scratch after the LP terminates for x_i. However, this is not valid for the LP in Equation 3.1. As the Lark's algorithm considers only the set of clean vectors, there is no guarantee that the termination point of the LP is on the skyline graph. After a new vector is added to the set of clean vectors, another routine should be called to find a basic feasible point for this new set of clean vectors.

Second difference is the objective function. While Lark's algorithm tries to find the maximum contribution of a new vector $\boldsymbol{\gamma}_i$ to the clean set (that is the minimum value of δ can be negative), the Skyline algorithm avoids finding the

greatest contribution of the $\boldsymbol{\gamma}_i$ vector to the sub-skyline graph, since all the vectors are already being considered and we are on the skyline graph at any iteration of the LP (x_i should be non-negative).

3.3 Mathematical Preliminaries for the Vector Pruning Problem

In the previous chapter, we have introduced two existing pruning algorithms from the literature. Although these algorithms succeed the pruning operation by writing different linear programs, the simplex tableaus written for these two algorithms are of a similar structure. In this chapter, we will have a closer look at the structure of the tableau and analyze what simplex iterations refer to.

First, note that the LP given in Equation 3.3 can be written in a different format in which every hyperplane equation is written on its own. For this, variable $y \geq b\boldsymbol{\gamma}_i, \forall i \in F_0$ is defined.

$$\begin{aligned} &\min x_i \\ &b\boldsymbol{\gamma}_j - y + x_j = 0, \quad \forall j \in F_0 \\ &\sum_{l=1}^{D} b(l) = 1 \\ &b(l) \geq 0, \quad \forall l \in \{1, \ldots, D\} \\ &x_j \geq 0, \quad \forall j \in \{1, \ldots, N\} \\ &y \geq 0 \end{aligned} \tag{3.4}$$

This modification has no impact on the feasible solutions of the coordinate vector b and slack variables x_j, but makes the manipulations and analysis on the LP easier. Note that $\boldsymbol{\gamma}_i \in \mathbb{R}_+^D$. Defining $\mathbf{H} = \begin{bmatrix} \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \ldots & \boldsymbol{\gamma}_N \end{bmatrix}^T$, and $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \ldots & x_N \end{bmatrix}^T$, Equation 3.4 is written in the matrix form;

$$\begin{aligned} &\min \mathbf{cx} \\ &\begin{bmatrix} \mathbf{H} & -\mathbf{e} & \mathbf{I} \\ \mathbf{e}^T & 0 & 0 \end{bmatrix} \begin{bmatrix} b^T \\ y \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \end{aligned} \tag{3.5}$$

where $\mathbf{0}$ is the zero vector, $\mathbf{e} = [1 \dots 1]^T$, and $\mathbf{I}$ is the identity matrix of relevant sizes. There are a total of $D + N + 1$ variables in the simplex tableau; D coordinate variables b, variable y and N slack variables $\mathbf{x}$. This is not the typical demonstration of an LP, as the slack variables x_i are also included in the variable vector. However, it is convenient to call $\mathbf{x}$ as the vector of slack variables, while it is multiplied by $\mathbf{I}$ in the simplex tableau for the inequality constraints. Moreover, the objective function of the linear program is also defined w.r.t $\mathbf{x}$. $\mathbf{c}$ is a zero row vector with a 1 in the ith entry. This indicates that the aim is to minimize the slack of the ith constraint.

For any of the $b\boldsymbol{\gamma}_i \leq y$ type of inequality constraints in Equation 3.5, if $x_i = 0$ then the constraint is satisfied as an equality. In linear programming literature, this is called as an active constraint. As explained in Section 3.2.2, it is possible to satisfy at most D of the $y \geq b\boldsymbol{\gamma}_i$ type of constraints to be active (discarding degeneracies).

Definition 11. *At any point of the simplex iterations, the active vectors from the set $\bar{\Gamma}$ are shown with Γ_A.*

There are two possible cases for $|\Gamma_A|$ that needs elaboration. We will discuss the case where $|\Gamma_A| = D$ in the following section and provide the case where $|\Gamma_A| < D$ in Appendix B.

3.3.1 Case Analysis for $|\Gamma_A| = D$

Without loss of generality, assume that $\Gamma_A = \{\boldsymbol{\gamma}_i\}_{i=1}^{D}$. We will partition matrix $\mathbf{H}$ into two parts; $\mathbf{H_A} = \begin{bmatrix} \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \dots & \boldsymbol{\gamma}_D \end{bmatrix}^T$ and $\mathbf{H_N} = \begin{bmatrix} \boldsymbol{\gamma}_{D+1} & \boldsymbol{\gamma}_{D+2} & \dots & \boldsymbol{\gamma}_N \end{bmatrix}^T$. We obtain the following simplex tableau that includes all of the constraints as well as the right hand side of the equations as the last column. The final row represents the objective function of the optimization where $\mathbf{c} = [\mathbf{c_A}\ \mathbf{c_N}]$.

$$\left[\begin{array}{ccccc} \mathbf{H_A} & -\mathbf{e} & \mathbf{I} & 0 & 0 \\ \mathbf{H_N} & -\mathbf{e} & 0 & \mathbf{I} & 0 \\ \mathbf{e}^T & 0 & 0 & 0 & 1 \\ \hline 0 & 0 & \mathbf{c_A} & \mathbf{c_N} & 0 \end{array}\right] \equiv \left[\begin{array}{ccccc} \mathbf{H_A} & -\mathbf{e} & 0 & \mathbf{I} & 0 \\ \mathbf{e}^T & 0 & 0 & 0 & 1 \\ \mathbf{H_N} & -\mathbf{e} & \mathbf{I} & 0 & 0 \\ \hline 0 & 0 & \mathbf{c_N} & \mathbf{c_A} & 0 \end{array}\right] \tag{3.6}$$

The second matrix in Equation 3.6 is obtained by changing the order of the 2nd and 3rd row blocks and third and the fourth column blocks. The sizes of the identity matrices are not same as well as the sizes of the zero matrices, but each can be found from the structure of the matrix. Discarding the objective function, LP multiplies the second matrix of Equation 3.6 by the inverse of the following part of it.

$$\mathbf{S} = \begin{bmatrix} \mathbf{H_A} & -\mathbf{e} & 0 \\ \mathbf{e}^T & 0 & 0 \\ \mathbf{H_N} & -\mathbf{e} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{S_A} & 0 \\ \mathbf{S_N} & \mathbf{I} \end{bmatrix} \tag{3.7}$$

where $\mathbf{S_A} = \begin{bmatrix} \mathbf{H_A} & -\mathbf{e} \\ \mathbf{e}^T & 0 \end{bmatrix}$ and $\mathbf{S_N} = \begin{bmatrix} \mathbf{H_N} & -\mathbf{e} \end{bmatrix}$. Note that $\mathbf{S}$ is invertible if $\mathbf{S_A}$ is invertible.

Lemma 3.3.1. *Take Γ_A. Assume that the set of equations*

$$\mathbf{S_A} \begin{bmatrix} b^T \\ y \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$$

has a solution $[b_A\ y_A]^T$. Then, $\mathbf{S_A}$ is invertible and the inverse is given in Equation 3.8.

Proof. We know that the vectors in $\mathbf{H_A}$ are selected to be linearly independent so it is invertible. Therefore the first D rows of $\mathbf{S_A}$ are linearly independent. For $\mathbf{S_A}$ to be singular, we should be able to write its final row in terms of the first D rows. Then we should be able to write,

$$\mathbf{h} \begin{bmatrix} \mathbf{H_A} & -\mathbf{e} \end{bmatrix} = \begin{bmatrix} \mathbf{e}^T & 0 \end{bmatrix}$$

which has a solution $\mathbf{h} = \mathbf{H_A}^{-1}\mathbf{e}$. Then using the solution $[b_A\ y_A]^T$,

$$[\mathbf{h}\ -1]\mathbf{S_A} \begin{bmatrix} b_A^T \\ y_A \end{bmatrix} = [\mathbf{h}\ -1] \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} = -1 \neq 0$$

Therefore, $\text{rank}(\mathbf{S_A}) = D + 1$ and $\mathbf{S_A}$ is invertible. Take $\alpha = \mathbf{e}^T\mathbf{H_A}^{-1}\mathbf{e}$. Using

matrix inversion property,

$$\mathbf{S_A}^{-1} = \begin{bmatrix} \mathbf{H_A}^{-1} - \alpha^{-1}\mathbf{h}\mathbf{e}^T\mathbf{H_A}^{-1} & \alpha^{-1}\mathbf{h} \\ -\alpha^{-1}\mathbf{e}^T\mathbf{H_A}^{-1} & \alpha^{-1} \end{bmatrix} \tag{3.8}$$

$$\begin{bmatrix} b_A^T \\ y_A \end{bmatrix} = \begin{bmatrix} \alpha^{-1}\mathbf{h} \\ \alpha^{-1} \end{bmatrix} \tag{3.9}$$

■

Recall that, the LP defined in Equation 3.4 has the objective function min x_i which is the slack variable of the hyperplane equation for vector $\boldsymbol{\gamma}_i$. After any vector $\boldsymbol{\gamma}_i$ is selected, we need to find the set of active vectors Γ_A from the set of dirty vectors $\bar{\Gamma}$ that minimizes x_i. The following lemma will formulate the problem in an inverse fashion; which vectors from the set $\bar{\Gamma}$ should be selected as the objective function for a given set of active vectors Γ_A? To answer this question, we will first introduce the following lemma.

Lemma 3.3.2. *In the simplex tableau given in Equation 3.6, the optimization procedure for any vector $\boldsymbol{\gamma}_i$ can be terminated if and only if the corresponding $b\boldsymbol{\gamma}_i - y + x_i = 0$ type of constraint row have all negative coefficients.*

Proof. Note that the objective function candidates are the slack variables x_i. Now suppose x_i belongs to one of the inactive vectors ($\mathbf{c_N} \neq 0$) in Equation 3.6. Then, we can write the simplex tableau as in Equation 3.10. It is known that the LP operations done in Equation 3.10 do not change the set of feasible solutions.

$$\left[\begin{array}{ccc} \mathbf{S_A} & 0 & \mathbf{I} \\ \mathbf{S_N} & \mathbf{I} & 0 \\ \hline 0 & \mathbf{c_N} & 0 \end{array}\right] \sim \left[\begin{array}{ccc} \mathbf{I} & 0 & \mathbf{S_A}^{-1} \\ 0 & \mathbf{I} & -\mathbf{S_N}\mathbf{S_A}^{-1} \\ \hline 0 & \mathbf{c_N} & 0 \end{array}\right] = \mathbf{P} \tag{3.10}$$

Assume that the nonzero entry of $\mathbf{c_N} = \mathbf{e_i}^T$ corresponds to the slack variable of $b\boldsymbol{\gamma}_i - y + x_i = 0$. This constraint is represented as one of the rows in the second row block of $\mathbf{P}$ in Equation 3.10. This row is in the form $\mathbf{r}_1 = \begin{bmatrix} \mathbf{0} & \mathbf{e_i}^T & -\tilde{\boldsymbol{\tau}}_i & x_i \end{bmatrix}$ where x_i is the value of the slack variable for this feasible solution.

The last row block of $\mathbf{P}$ in Equation 3.10 which corresponds to the objective function is in the form $\mathbf{r}_2 = \begin{bmatrix}\mathbf{0} & \mathbf{e_i}^T & \mathbf{0} & 0\end{bmatrix}$. The LP operation at this step requires the subtraction of $\mathbf{r}_1$ from $\mathbf{r}_2$. Therefore $\tilde{\boldsymbol{\tau}}_i \geq 0$, then the resultant coefficients in the objective function will be all non-negative, which means the end of the optimization for the objective function min x_i. The same idea is valid for any other row corresponding to the second row block of P. ■

Lemma 3.3.2 shows that for any selection of Γ_A, by only doing a simple sign check of the block matrix $\mathbf{S_N S_A}^{-1}$, we can see if the inactive vectors are dominated for a particular selection of Γ_A. We will demonstrate this with an example.

Table 3.1: Checking all vectors in $\bar{\Gamma}$, $D = 2$

	$b(1)$	$b(2)$	y	x_1	x_2	x_3	x_4	RHS
$\boldsymbol{\gamma}_1$	4	0	-1	1	0	0	0	0
$\boldsymbol{\gamma}_2$	0	4	-1	0	1	0	0	0
$\boldsymbol{\gamma}_3$	2	1.9	-1	0	0	1	0	0
$\boldsymbol{\gamma}_4$	1	2.9	-1	0	0	0	1	0
SC	1	1	0	0	0	0	0	1
z	0	0	0	0	0	1	0	0
				⇓				
	$b(1)$	$b(2)$	y	x_3	x_4	x_1	x_2	RHS
$\boldsymbol{\gamma}_1$	1	0	0	0	0	0.125	-0.125	0.50
$\boldsymbol{\gamma}_2$	0	1	0	0	0	-0.125	0.125	0.50
SC	0	0	1	0	0	-0.500	-0.500	2.00
$\boldsymbol{\gamma}_3$	0	0	0	1	0	-0.513	-0.488	0.05
$\boldsymbol{\gamma}_4$	0	0	0	0	1	-0.263	-0.738	0.05
z	0	0	0	0	0	0.513	0.488	-0.05

Suppose that the dirty set of vectors is $\bar{\Gamma} = \{[4\ 0]^T, [0\ 4]^T, [2\ 1.9]^T, [1\ 2.9]^T\}$. Clearly, none of these vectors are pointwise dominated by another. In Table 3.1, the hyperplane equations for each vector are written in the first four rows and the fifth row corresponds to the simplex constraint. Assume that we are trying to minimize the slack variable x_3.

The final tableau is also given in Table 3.1. As can be seen here, $x_3 = 0.05 > 0$ in the optimal point so $\boldsymbol{\gamma}_3$ is not in the clean set. Moreover, $\boldsymbol{\gamma}_4$ has also reached the optimal point as the coefficients of the non-basic variables are all negative ($-\tilde{\boldsymbol{\tau}}_4 = [-0.263\ \ -0.738]$) and $x_4 = 0.05 > 0$. Therefore, both of the hyperplane equations for $\boldsymbol{\gamma}_3$ and $\boldsymbol{\gamma}_4$ will be erased from the table in the following round.

Lemma 3.3.2 exploits the multiple objective function idea. For any selection of the active vectors Γ_A, we are trying to find the inactive vectors from the set $\bar{\Gamma}$, such that $\tilde{\boldsymbol{\tau}}_i \geq 0$. As Γ_A is already selected in the simplex tableau, the inactive vectors are now treated not as constraints but as objective function candidates.

Based on this argument, for each Γ_A set, small sized special LPs are generated. The constraints of any of them are formed from $\Gamma_A \cup \{\boldsymbol{\gamma}_i\}$ where $\boldsymbol{\gamma}_i \in \bar{\Gamma} \setminus \Gamma_A$ and the objective function is the slack variable of $\boldsymbol{\gamma}_i$, so this LP ends up with the conclusion about the domination or non-domination of $\boldsymbol{\gamma}_i$ by the vectors in Γ_A. This LP is named as $LP_{A,i}$ where A denotes the set Γ_A and i is the index of $\boldsymbol{\gamma}_i$. Using the notation from Lemma 3.3.2 and taking $i = 0$, we define;

$$\mathbf{S_N} = [\boldsymbol{\gamma}_0^T \ \ -1]$$
$$\mathbf{S_N}{\mathbf{S_A}}^{-1} = \left[(\boldsymbol{\gamma}_0^T + \alpha^{-1}(1 - \boldsymbol{\gamma}_0^T\mathbf{h})\mathbf{e}^T){\mathbf{H_A}}^{-1} \quad \alpha^{-1}(\boldsymbol{\gamma}_0^T\mathbf{h} - 1)\right]$$

Then,

$$x_0 = \alpha^{-1}(1 - \boldsymbol{\gamma}_0^T\mathbf{h}) \tag{3.11}$$
$$\tilde{\boldsymbol{\tau}}_0 = (\boldsymbol{\gamma}_0^T + x_0\mathbf{e}^T){\mathbf{H_A}}^{-1} \tag{3.12}$$

Using Equation 3.11 and 3.9, we can write

$$x_0 = \alpha^{-1} - \alpha^{-1}\mathbf{h}^T\boldsymbol{\gamma}_0 = y_A - b_A\boldsymbol{\gamma}_0 \tag{3.13}$$

Definition 12. *Define* $\tilde{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0 + x_0\mathbf{e}$ *where* x_0 *is given in Equation 3.11. Note that* $\tilde{\boldsymbol{\gamma}}_0$ *depends on* Γ_A.

With Definition 12, symbol $\tilde{\boldsymbol{\tau}}_0$ in Equation 3.12 becomes the vector of coefficients for $\tilde{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0 + x_0\mathbf{e}$ when it is written as a linear combination of the vectors in Γ_A. Lemma 3.3.3 shows another property of these vector of coefficients, $\tilde{\boldsymbol{\tau}}_0$.

Lemma 3.3.3. $\tilde{\boldsymbol{\tau}}_0\mathbf{e} = 1$.

Proof.

$$\begin{aligned}\tilde{\boldsymbol{\tau}}_0\mathbf{e} &= (\boldsymbol{\gamma}_0^T + x_0\mathbf{e}^T){\mathbf{H_A}}^{-1}\mathbf{e} \\ &= \boldsymbol{\gamma}_0^T\mathbf{h} + x_0\alpha \\ &= \boldsymbol{\gamma}_0^T\mathbf{h} + (1 - \boldsymbol{\gamma}_0^T\mathbf{h}) \\ &= 1\end{aligned}$$

■

As discussed in Lemma 3.3.2, for a given Γ_A, we will select $\boldsymbol{\gamma}_0$ such that the corresponding $\tilde{\boldsymbol{\tau}}_0$ has non-negative elements.

3.3.2 Convexity Analysis for $|\Gamma_A| = D$

The definition of a convex cone is as follows:

Definition 13. *The convex cone formed by Γ_A is described by*

$$\Gamma_A^C = \left\{ \boldsymbol{\gamma}_0 \in \mathbb{R}^D : \quad \boldsymbol{\gamma}_0 = \sum_{\boldsymbol{\gamma}_i \in \Gamma_A} \boldsymbol{\tau}_0(i)\boldsymbol{\gamma}_i,\ \boldsymbol{\tau}_0(i) \geq 0 \right\}$$

Note that the above definition is equivalent to $\boldsymbol{\gamma}_0 = {\mathbf{H_A}}^T \boldsymbol{\tau}_0^T$ where $\boldsymbol{\tau}_0 = [\boldsymbol{\tau}_0(1) \dots \boldsymbol{\tau}_0(D)]$.

The following theorem is equivalent to Lemma 3.3.2 but considers the convex cone Γ_A^C.

Theorem 3.3.4. *Take Γ_A. The following statement is true:*

$$\left(\tilde{\boldsymbol{\gamma}}_0 \in \Gamma_A^C\right) \wedge (x_0 > 0) \Longrightarrow R(\boldsymbol{\gamma}_0, \Gamma_A) = \emptyset$$

Proof. Take any $b \in \mathcal{B}$. We can write $b = b_A + \nabla b$ such that $\nabla b \mathbf{e} = 0$. Select $k = \arg\max_{\boldsymbol{\gamma}_i \in \Gamma_A} \nabla b \boldsymbol{\gamma}_i$. Then

$$\begin{aligned}
b\boldsymbol{\gamma}_0 &= (b_A + \nabla b)\boldsymbol{\gamma}_0 \\
&= b_A\boldsymbol{\gamma}_0 + \nabla b\boldsymbol{\gamma}_0 \\
&= y_A - x_0 + \nabla b\tilde{\boldsymbol{\gamma}}_0 \\
&< y_A + \nabla b\tilde{\boldsymbol{\gamma}}_0 \\
&= y_A + \sum_{i=1}^{D} \tilde{\boldsymbol{\tau}}_0(i)(\nabla b\boldsymbol{\gamma}_i) \\
&< y_A + \nabla b\boldsymbol{\gamma}_k \\
&= (b_A + \nabla b)\boldsymbol{\gamma}_k \\
&= b\boldsymbol{\gamma}_k
\end{aligned}$$

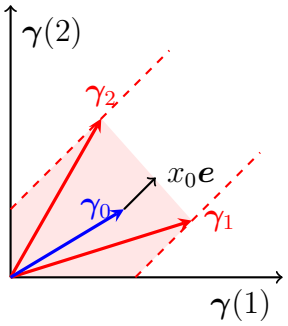


Figure 3.2: Supplementary Figure for Theorem 3.3.4. $\Gamma_A = \{\boldsymbol{\gamma}_i\}_{i=1}^2$ and $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma_A^C$ and $x_0 > 0$, therefore $\boldsymbol{\gamma}_0$ is dominated.

■

Theorem 3.3.4 shows that if $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma_A^C$ and $x_0 \geq 0$, then the optimization is finished. The following theorem will prove a complementary fact; if $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma_A^C$ and $x_0 < 0$, there is a procedure of selecting another set Γ_{A_t} such that $\tilde{\boldsymbol{\gamma}}_0^t \in \Gamma_{A_t}^C$ and the corresponding x_0^t is increased. To cope with multiple A sets, we modify the notation by introducing the subscript t.

Definition 14. *If there are more than one set of vectors Γ_{A_t}, then $\tilde{\boldsymbol{\gamma}}_0^t = \boldsymbol{\gamma}_0 + x_0^t \mathbf{e}$. Similarly, $\boldsymbol{\gamma}_0 = \mathbf{H_{A_t}}^T (\boldsymbol{\tau}_0^t)^T$ and $\tilde{\boldsymbol{\gamma}}_0^t = \mathbf{H_{A_t}}^T (\tilde{\boldsymbol{\tau}}_0^t)^T$. The set Γ_{A_t} can be equally represented by $A_t = \{j : \boldsymbol{\gamma}_j \in \Gamma_{A_t}\}$*

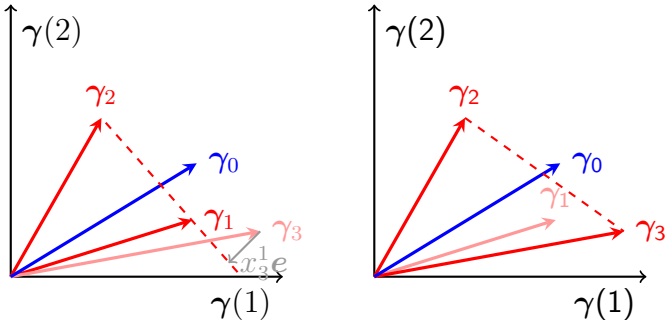


Figure 3.3: Supplementary Figure for Theorem 3.3.5. $\Gamma_{A_1} = \{\boldsymbol{\gamma}_i\}_{i=1}^2$ and $\tilde{\boldsymbol{\gamma}}_0^1 \in \Gamma_{A_1}^C$. Note that $x_3^1 < 0$ and it is not necessary that $\tilde{\boldsymbol{\gamma}}_3^1 \in \Gamma_{A_1}^C$. Select $\Gamma_{A_2} = \{\boldsymbol{\gamma}_i\}_{i=2}^3$. Then, $x_0^2 > x_0^1$

Theorem 3.3.5. *Take $\Gamma_{A_1} = \{\boldsymbol{\gamma}_j\}_{j=1}^D$. Let $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_{D+1}$ be such that $\tilde{\boldsymbol{\gamma}}_0^1 \in \Gamma_{A_1}^C$ and $x_{D+1}^1 < 0$. Then, there is another set of vectors $\Gamma_{A_2} = \{\boldsymbol{\gamma}_j\}_{j=1, j\neq k}^{D+1}$ such that $\tilde{\boldsymbol{\gamma}}_0^2 \in \Gamma_{A_2}^C$ and $x_0^2 > x_0^1$. Moreover, this set Γ_{A_2} is unique for a given vector $\boldsymbol{\gamma}_0$.*

Proof. For $l \in \{0, D+1\}$, we can write $\tilde{\boldsymbol{\gamma}}_l^1 = \sum_{j=1}^{D} \tilde{\boldsymbol{\tau}}_l^1(j)\boldsymbol{\gamma}_j$ and $\tilde{\boldsymbol{\tau}}_l^1\mathbf{e} = 1$. Moreover, $\tilde{\boldsymbol{\tau}}_0^1 > 0$.

We know that $\tilde{\boldsymbol{\gamma}}_{D+1}^1 = \sum_{j=1}^{D} \tilde{\boldsymbol{\tau}}_{D+1}^1(j)\boldsymbol{\gamma}_j$. Then for any vector $\boldsymbol{\gamma}_k \in \Gamma_{A_1}$, we can write,

$$\boldsymbol{\gamma}_k = \frac{1}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\tilde{\boldsymbol{\gamma}}_{D+1}^1 - \sum_{j=1, j\neq k}^{D} \frac{\tilde{\boldsymbol{\tau}}_{D+1}^1(j)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\boldsymbol{\gamma}_j \tag{3.14}$$

Selecting $k = \arg\min_{j:\tilde{\boldsymbol{\tau}}_{D+1}^1(j)>0} \frac{\tilde{\boldsymbol{\tau}}_0^1(j)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(j)}$, we have

$$\tilde{\boldsymbol{\gamma}}_0^1 = \tilde{\boldsymbol{\tau}}_0^1(k)\boldsymbol{\gamma}_k + \sum_{j=1, j\neq k}^{D} \tilde{\boldsymbol{\tau}}_0^1(j)\boldsymbol{\gamma}_j \tag{3.15}$$

$$= \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\tilde{\boldsymbol{\gamma}}_{D+1}^1 + \sum_{j=1, j\neq k}^{D} \left(\tilde{\boldsymbol{\tau}}_0^1(j) - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)\tilde{\boldsymbol{\tau}}_{D+1}^1(j)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\right)\boldsymbol{\gamma}_j \tag{3.16}$$

Define,

$$\tilde{\boldsymbol{\tau}}_0^2(j) = \begin{cases} \tilde{\boldsymbol{\tau}}_0^1(j) - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)\tilde{\boldsymbol{\tau}}_{D+1}^1(j)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)} & 1 \leq j \leq D,\ j \neq k \\ \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)} & j = D+1 \end{cases}$$

Recall that $\tilde{\boldsymbol{\gamma}}_{D+1}^1 = \boldsymbol{\gamma}_{D+1} + x_{D+1}^1\mathbf{e}$. Inserting this equation into Equation 3.16;

$$\tilde{\boldsymbol{\gamma}}_0^1 - \tilde{\boldsymbol{\tau}}_0^2(D+1)x_{D+1}^1\mathbf{e} = \sum_{j=1, j\neq k}^{D+1} \tilde{\boldsymbol{\tau}}_0^2(j)\boldsymbol{\gamma}_j \tag{3.17}$$

Note that $\tilde{\boldsymbol{\tau}}_0^2(j) = \tilde{\boldsymbol{\tau}}_{D+1}^1(j)\left(\frac{\tilde{\boldsymbol{\tau}}_0^1(j)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(j)} - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\right) > 0,\ j \neq D+1$. From selection $\tilde{\boldsymbol{\tau}}_{D+1}^1(k) \geq 0$. Therefore $\tilde{\boldsymbol{\tau}}_0^2 \geq 0$. Moreover,

$$\sum_{j=1, j\neq k}^{D+1} \tilde{\boldsymbol{\tau}}_0^2(j) = \sum_{j=1}^{D} \tilde{\boldsymbol{\tau}}_0^1(j) - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}\left(\sum_{j=1}^{D} \tilde{\boldsymbol{\tau}}_{D+1}^1(j) - 1\right) \tag{3.18}$$

$$= 1$$

Equation 3.18 and Lemma 3.3.3 show that $\tilde{\boldsymbol{\gamma}}_0^2 = {\mathbf{H}_{\mathbf{A}_2}}^T\tilde{\boldsymbol{\tau}}_0^2$. Then we can write,

$$\tilde{\boldsymbol{\gamma}}_0^2 = \tilde{\boldsymbol{\gamma}}_0^1 - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}x_{D+1}^1\mathbf{e} = \boldsymbol{\gamma}_0 + x_0^1\mathbf{e} - \frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)}x_{D+1}^1\mathbf{e} \tag{3.19}$$

$$= \boldsymbol{\gamma}_0 + x_0^2\mathbf{e} \tag{3.20}$$

We know that $x_{D+1}^1 < 0$. Then, $x_0^2 > x_0^1$.

Now we will prove that the selection of k is unique. Assume that we have selected k' instead of k. Then, $\tilde{\boldsymbol{\tau}}_0^2(k) = \tilde{\boldsymbol{\tau}}_{D+1}^1(k)\left(\frac{\tilde{\boldsymbol{\tau}}_0^1(k)}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k)} - \frac{\tilde{\boldsymbol{\tau}}_0^1(k')}{\tilde{\boldsymbol{\tau}}_{D+1}^1(k')}\right) < 0$. ■

Recall from Theorem 3.3.4 and Definition 14, for a given vector $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma^C_{A_t}$, $x_0^t \geq 0$ means that vector $\boldsymbol{\gamma}_0$ is dominated. Theorem 3.3.5 guarantees that at every step t, vector $\boldsymbol{\gamma}_0$ remains in a convex cone Γ_{A_t} and we are increasing the value of x_0^t as shown in Equation 3.19. This is done by replacing one of the vectors in Γ_{A_t} with a vector $\boldsymbol{\gamma}_{D+1}$ such that $x_{D+1}^t < 0$. Note it is not obligatory that $\tilde{\boldsymbol{\gamma}}_{D+1}^t \in \Gamma^C_{A_t}$.

Theorem 3.3.5 actually writes the constraint $b\boldsymbol{\gamma}_{D+1} - y + x_{D+1} = 0$ into $LP_{A_1,0}$ where A_1 denotes the initial active set Γ_{A_1} and 0 is the index of $\boldsymbol{\gamma}_0$. While $x_{D+1}^1 < 0$, this newly added constraint violates the basic feasibility of the simplex tableau. The procedure described in the theorem is one step of the dual simplex algorithm where primal feasibility is again satisfied.

Now, we need to show that the procedure defined in Theorem 3.3.5 ends up in a unique solution. For this, we will assume that we have selected vector $\boldsymbol{\gamma}_0$ and started from different initial convex regions to apply Theorem 3.3.5. Theorem 3.3.6 shows that the final convex region is unique.

Theorem 3.3.6. *Take any A_1, A_2. Suppose that*

$$\mathbf{S_{A_t}} \begin{bmatrix} b_t^T \\ \\ y_t \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \\ 1 \end{bmatrix} \quad t \in \{1, 2\}$$

has solutions such that $b_1 \neq b_2$. Moreover, $\forall \boldsymbol{\gamma}_i \in \Gamma_{A_1} \setminus \Gamma_{A_2}$, $b_2\boldsymbol{\gamma}_i < y_2$ and $\forall \boldsymbol{\gamma}_i \in \Gamma_{A_2} \setminus \Gamma_{A_1}$, $b_1\boldsymbol{\gamma}_i < y_1$. Given these conditions, the following statement is true for any $\boldsymbol{\gamma}_0$:

$$\tilde{\boldsymbol{\gamma}}_0^1 \in \Gamma^C_{A_1} \implies x_0^1 < x_0^2 \text{ and } \tilde{\boldsymbol{\gamma}}_0^2 \notin \Gamma^C_{A_2}$$

Proof. We can write for $t \in \{1, 2\}$,

$$\boldsymbol{\gamma}_0 + x_0^t \mathbf{e} = \sum_{j \in A_t} \tilde{\boldsymbol{\tau}}_0^t(j) \boldsymbol{\gamma}_j \tag{3.21}$$

where $\tilde{\boldsymbol{\tau}}_0^1 > 0$. From Definition 14, $b_2\boldsymbol{\gamma}_0 + x_0^2 = y_2$. Write,

$$b_2\boldsymbol{\gamma}_0 + x_0^1 = \sum_{j\in A_1} \tilde{\boldsymbol{\tau}}_0^1(j) b_2\boldsymbol{\gamma}_j \tag{3.22}$$

$$= \sum_{j\in A_1\cap A_2} \tilde{\boldsymbol{\tau}}_0^1(j) b_2\boldsymbol{\gamma}_j + \sum_{j\in A_1\setminus A_2} \tilde{\boldsymbol{\tau}}_0^1(j) b_2\boldsymbol{\gamma}_j \tag{3.23}$$

$$= \sum_{j\in A_1\cap A_2} \tilde{\boldsymbol{\tau}}_0^1(j) y_2 + \sum_{j\in A_1\setminus A_2} \tilde{\boldsymbol{\tau}}_0^1(j) b_2\boldsymbol{\gamma}_j \tag{3.24}$$

$$< \sum_{j\in A_1} \tilde{\boldsymbol{\tau}}_0^1(j) y_2 = y_2 \tag{3.25}$$

Then $x_0^1 < x_0^2$. Now suppose that $\tilde{\boldsymbol{\gamma}}_0^2 \in \Gamma_{A_2}^C$. Writing the same equations for $b_1\tilde{\boldsymbol{\gamma}}_0^1$ and $b_1\tilde{\boldsymbol{\gamma}}_0^2$ we would arrive at $x_0^2 < x_0^1$. This is a contradiction. Therefore $\tilde{\boldsymbol{\gamma}}_0^2 \notin \Gamma_{A_2}^C$. ■

Combined with Theorem 3.3.6, it can be asserted that if a convex cone Γ_{A_t} is found such that the intersection point of the hyperplanes $[b_t\ y_t]^T$ is on the skyline, then the optimization for $\boldsymbol{\gamma}_0$ is finished. If $b_t\boldsymbol{\gamma}_0 \neq y_t$, vector $\boldsymbol{\gamma}_0$ is dominated.

3.4 FastCone Algorithm

Algorithm 4 is the FastCone Algorithm. The main routine FC is basically the initialization and utilization of the simplex tableau $\mathbf{P}$. The subroutines used by FC are provided by another table. At the initialization phase of the algorithm, F and Q are set as the current dirty and clean vector indices, respectively. Another index set, A, consists of the indices of the clean vectors inside $\mathbf{P}$. Therefore, at any time $A \subset Q$.

The algorithm starts with LPINIT, which initializes the simplex tableau $\mathbf{P}$. The tableau is initialized by using vector $\boldsymbol{\gamma}_i = w(b, \bar{\Gamma})$ where $b(1) = 1$. For the objective function we select the vector $\boldsymbol{\gamma}_j$ where $j = \arg\min_{k\in F} b\boldsymbol{\gamma}_k$. The constraint defined by the dirty vector $\boldsymbol{\gamma}_j$ is not written to $\mathbf{P}$, therefore x_j is not defined. The aim of the algorithm is to find a convex cone Γ_A where $\boldsymbol{\gamma}j$ is eliminated. Therefore we narrow down the convex cone around $\boldsymbol{\gamma}_j$. If the algorithm finally arrives at the best possible convex cone which contains vector $\boldsymbol{\gamma}_j$ and no other selection of the set Γ_A is possible, we conclude that vector

Algorithm 4 FastCone Algorithm

procedure FC($\bar{\Gamma}$)
  $Q \leftarrow \emptyset$, $A \leftarrow \emptyset$, $F \leftarrow F_0$
  $b_1(1) = 1, i \leftarrow \arg\max_{k \in F} b_1\gamma_k$, $j \leftarrow \arg\min_{k \in F} b_1\gamma_k(1)$
  $F \leftarrow F \setminus \{i\}$, $Q \leftarrow \{i\}$, $A \leftarrow \{i\}$
  $\mathbf{H} = \begin{bmatrix} \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \dots & \boldsymbol{\gamma}_N \end{bmatrix}$
  $\mathbf{X} = zeros(\text{NSTATSMAX}, |F|)$
  $\boldsymbol{\Delta}\mathbf{X} = zeros(\text{NSTATSMAX}, |F|)$
  $\mathbf{P} \leftarrow \text{LPINIT}(\gamma_i, \gamma_j)$
  $t \leftarrow 1$, $\mathbf{X}(t,:) = b_t\mathbf{H} - y_t\mathbf{e}$
  while $F \neq \emptyset$ **do**
    while $\mathbf{P}$ not optimum **do**
      $t \leftarrow t + 1$
      $(b_t, y_t, \mathbf{P}) \leftarrow \text{LPITER1}(\mathbf{P})$
      $\mathbf{X}(t,:) = b_t\mathbf{H} - y_t\mathbf{e}$
      $\boldsymbol{\Delta}\mathbf{X}(t,:) = \mathbf{X}(t,:) - \mathbf{X}(t-1,:)$
    end while
    while $\mathbf{X}(t,j) > 0 \wedge j \neq \arg\min_k \mathbf{X}(t,k)$ **do**
      $i \leftarrow \arg\min_k \mathbf{X}(t,k)$
      $F \leftarrow F \setminus \{i\}$, $Q \leftarrow Q \cup \{i\}$, $A \leftarrow A \cup \{i\}$
      add $b\gamma_i + x_i - y = 0$ to $\mathbf{P}$
      while $\mathbf{P}$ not feasible **do**
        $t \leftarrow t + 1$
        $(b_t, y_t, \mathbf{P}) \leftarrow \text{LPITER2}(\mathbf{P})$
        $\mathbf{X}(t,:) = b_t\mathbf{H} - y_t\mathbf{e}$
        $\boldsymbol{\Delta}\mathbf{X}(t,:) = \mathbf{X}(t-1,:) - \mathbf{X}(t,:)$
      end while
      if $|A| >$ NVMAX **then**
        $A_t \leftarrow$ active vectors inside $\mathbf{P}$
        $d_0 = 0$
        for all $k \in A \setminus A_t$ **do**
          $d = \text{sum}(\boldsymbol{\Delta}\mathbf{X}(t - \text{TMAX} : t, k) < 0)$
          if $d > d_0$ **then**
            $d_0 = d$, $i \leftarrow k$
          end if
        end for
        delete $b\gamma_i + x_i - y = 0$ from $\mathbf{P}$
        $A \leftarrow A - \{i\}$
      end if
    end while
    if $\mathbf{X}(t,j) < 0$ **then**
      $F \leftarrow F \setminus \{j\}$
      delete $b\gamma_j + x_j - y = 0$ from $\mathbf{P}$
    end if
    $j \leftarrow \text{LPOBJSLCT}(\boldsymbol{\Delta}\mathbf{X}, F)$
    set objective function of $\mathbf{P}$ to $\min y - b\boldsymbol{\gamma}_j$
  end while
  return $\bar{\Gamma}_Q$
end procedure

Subroutines for the FastCone Algorithm

procedure LPINIT($\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_j$)
write the initial simplex tableau $\mathbf{P}$
$\min y - b\boldsymbol{\gamma}_j$ subject to
$b\boldsymbol{\gamma}_i + x_i - y = 0$
$\sum_{l=1}^{D} b(l) = 1$
$b \geq 0, y > 0, x_i \geq 0$
return $\mathbf{P}$
end procedure

procedure LPITER1($\mathbf{P}$)
do one primal simplex iteration to $\mathbf{P}$
return $(b_t, y_t, \mathbf{P})$
end procedure

procedure LPITER2($\mathbf{P}$)
do one dual simplex iteration to $\mathbf{P}$
return $(b_t, y_t, \mathbf{P})$
end procedure

procedure LPOBJSLCT($\boldsymbol{\Delta}\mathbf{X}, F$)
$d_0 = 0$
for all $k \in F$ **do**
$d = \text{sum}(\boldsymbol{\Delta}\mathbf{X}(t - \text{TMAX} : t, k) > 0)$
if $d > d_0$ **then**
$d_0 = d$, $j \leftarrow k$
end if
end for
return j
end procedure

$\boldsymbol{\gamma}_j$ is not dominated. The objective function is taken as $\min y - b\boldsymbol{\gamma}_j$. Define $x_j := y - b\boldsymbol{\gamma}_j$. Note that x_j is not a slack variable while the constraint related to $\boldsymbol{\gamma}_j$ is not written inside the simplex tableau. This vector is used to determine the objective function.

NSTATSMAX is an estimated upper bound for the number of possible simplex iterations that should be determined before the algorithm starts. NVMAX determines the maximum number of vectors in the simplex tableau. All vectors inside the simplex tableau are checked at each step of the FastCone algorithm; therefore a higher number of vectors inside the simplex tableau increases the chance of dirty vectors to be eliminated.

The backbones of the algorithm are LPITER1 and LPITER2. LPITER1 is a primal simplex iteration and LPITER2 is a dual simplex iteration. To explain the operation of these two simplex iteration procedures, suppose that dirty vector $\boldsymbol{\gamma}_j$ is selected at time $t = t_0$ and the objective function is written as $\min y - b\boldsymbol{\gamma}_j$. Assume that the set of clean vectors inside $\mathbf{P}$ is A. We will define $x_j := y - b\boldsymbol{\gamma}_j$ again to stress that x_j is not a slack variable inside the simplex tableau. Initially, LPITER1 is repeated until a convex region Γ_{A_t} is found such that $\tilde{\boldsymbol{\gamma}}_j^t \in \Gamma_{A_t}^C$. Call this time as $t = t_1$. From Theorem 3.3.6, it is known that this convex region is unique for any given set of clean vectors A and $x_j^{t_1} < x_j^{t_0}$. Throughout these iterations, no vectors are added to the simplex tableau. If $x_j^{t_1} < 0$, vector $\boldsymbol{\gamma}_j$ can deleted from the dirty set of vectors and we can select another objective function.

If $x_j^{t_1} \geq 0$, $\boldsymbol{\gamma}_i = w(b^{t_1}, \bar{\Gamma})$ is selected and $b\boldsymbol{\gamma}_i - y + x_i = 0$ is added to the simplex tableau $\mathbf{P}$. Note that we can safely add vector $\boldsymbol{\gamma}_i$ to the simplex tableau as it is a non-dominated vector. When vector $\boldsymbol{\gamma}_i$ is added to the simplex tableau $\mathbf{P}$, primal feasibility of the solution is violated while $x_i^{t_1} < 0$ as described in Theorem 3.3.5. Note that there is a non-negativity constraint on the slack variables of the clean vectors. To solve this problem, LPITER2 is repeated until the primal feasibility of $\mathbf{P}$ is again satisfied. Call the time as $t = t_2$ where the primal feasibility is satisfied. Then, we have reached another cone $\Gamma_{A_{t_2}}$ such that $\tilde{\boldsymbol{\gamma}}_j^{t_2} \in \Gamma_{A_{t_2}}^C$. Note that $x_j^{t_2} < x_j^{t_1} < x_j^{t_0}$. This operation is repeated for the dirty vector $\boldsymbol{\gamma}_j$ until

there are no vectors left to add to the clean set which would decrease the slack variable x_j. Vector $\boldsymbol{\gamma}_j$ is eliminated as soon as $x_j < 0$. Therefore when x_j is optimum, there are only two possibilities; either $\boldsymbol{\gamma}_j \in \Gamma_{A_{t_2}}$ becomes an active vector and therefore $x_j = 0$, or $x_j > 0$ which shows that we have arrived at a convex cone which has $\boldsymbol{\gamma}_j$ as the only vector inside. Therefore vector $\boldsymbol{\gamma}_j$ is not dominated. After the decision about $\boldsymbol{\gamma}_j$ is given, a new dirty vector is selected using LPOBJSLCT procedure.

Notice that at any of these iterations, the values of the slack variables are stored in a matrix $\mathbf{X}$. $\boldsymbol{\Delta}\mathbf{X}$ shows the simultaneous decrease/increase in the slack variables of two vectors which is a good measure for their 'closeness'. It would show that the same pivoting operation is a valid step for the minimization/maximization of both of their slack variables. The algorithm checks for the similarity in the decrease/increase of the slack variables for two vectors in the last TMAX iterations by using matrix $\boldsymbol{\Delta}\mathbf{X}$ and decide on both the dirty vector $\boldsymbol{\gamma}_j$ and the set of clean vectors inside the simplex tableau, $\mathbf{P}$.

In the pseudocode, we have provided the matrices $\mathbf{X}$ and $\boldsymbol{\Delta}\mathbf{X}$ holding the information about the slack variables. It is also possible to write all vectors inside the simplex tableau and use the revised simplex algorithm which would have the same computational complexity. For such a simplex tableau the simultaneous increase and decrease in the $\boldsymbol{\Delta}\mathbf{X}$ for any two vectors would refer to a column check for these vectors in the simplex tableau. As these two representations are equivalent when the computational complexity is considered, we have selected to write $\mathbf{X}$ and $\boldsymbol{\Delta}\mathbf{X}$ as two matrices which is easier to think with the geometric framework provided in this study.

3.4.1 Comparison of FastCone algorithm to the conventional algorithms

Before going into the experimental results, we would like to give an analytical comparison of the FC algorithm to the conventional algorithms and their revised versions offered by us.

After a dirty vector $\boldsymbol{\gamma}_0$ from $\bar{\Gamma}$ is selected, the pruning algorithms come up with an active set of vectors, Γ_A, that shows whether this particular vector $\boldsymbol{\gamma}_0$ is dominated or not. Skyline algorithm finds an active set of vectors Γ_A that has a solution $[b_A \; y_A]^T$ on the skyline. However, if $x_0 >> 0$, some other selection of the Γ_A would also be enough to conclude that $\boldsymbol{\gamma}_0$ is dominated. Lark's algorithm selects a dirty vector $\boldsymbol{\gamma}_0$ as its objective function and adds clean vectors to the simplex tableau until the selection of Γ_A is sufficient to give a decision about $\boldsymbol{\gamma}_0$. For Lark's algorithm, the optimization could end at a convex region which may also contain other dominated vectors, but this fact is not exploited. While at every step, one clean vector is added to the simplex tableau, the LP is calculated from scratch.

In a former paper [36], we have discussed revisions for these two pruning algorithms. These revised algorithms are also provided at Appendix A. The revision offered for the Iterative Skyline Algorithm (ISKY) is based on the idea given in Lemma 3.3.2. In ISKY, all the constraints are added to the simplex tableau from the beginning. From Lemma 3.3.2, it is known that checking the optimality condition for the inactive vectors inside the simplex tableau is equivalent to making a sign check to the block matrix $\mathbf{S_N S_A^{-1}}$. The Iterative Skyline Algorithm with Multiple Objective Functions (ISwM) discussed in Section A.1 checks the new value of block matrix $\mathbf{S_N S_A^{-1}}$ at every simplex iteration and decides which of the vectors are dominated at this vertex. The pseudocode of the algorithm is presented in Algorithm 11.

The revision offered for the Lark's Algorithm is a heuristic to start the LP close to the optimal feasible point. For this, the vector distance between the the dirty vector and the clean vectors inside the simplex tableau, $||\boldsymbol{\gamma}_0 - \boldsymbol{\gamma}_i||$, is taken into account. The witness point to the clean vector $\boldsymbol{\gamma}_i$ is used to find an initial basic feasible point for the LP. This algorithm is called Lark's Algorithm with Initial Condition (LKwI). The revision idea and the algorithm is provided at section A.2. The pseudocode of the algorithm can be found in Algorithm 12.

FastCone algorithm combines the advantage of both of the revised algorithms. The algorithm tries to eliminate the dirty vector $\boldsymbol{\gamma}_0$ as soon as a set of active

vectors Γ_A is found such that $x_0 \approx 0$. Moreover, the statistics about the slack variables are used as a powerful vector selection criteria compared to the vector distance. As shown in Figure 3.2, the optimization for two vectors with different vector norms can end in the same convex region Γ_A^C. Without doing the pivoting operation inside the LP, by calculating the slack variables for the dirty vectors at each turn, we can determine for which of the dirty vectors the visited vertices are also a valid path for the minimization.

3.5 Simulations

In this section, we will present the result of the experiments with artificially created vectors and benchmark problems. FastCone Algorithm (FC) is compared to Lark's Algorithm (LRK), Iterative Skyline Algorithm (ISKY) and their revised versions; Lark's Algorithm with Initial Conditions (LRwI) and Iterative Skyline with Multi-Objective Functions (ISwM). The revised algorithms can be found in Appendix A. The results are provided in the tables and figures below. All of the pruning algorithms and the exact value iteration algorithm are implemented in MATLAB environment. The tests are performed with a standard desktop computer (Intel Core i7-3770 3.4GHz 8GB RAM).

3.5.1 Pruning Performance of Randomly Generated Sets

To demonstrate the scalability of the proposed algorithms, we have first tested them with artificial problems, as it gives the user the flexibility to design the problem parameters as needed. We first constructed a set of random vector sets $\{\Gamma_1, \ldots, \Gamma_M\}$. Random vectors in Γ_i are created by selecting D random numbers uniformly distributed between $(0, 200)$. Then, additional random vectors are generated and added to the set provided that they are not pointwise dominated. This process is repeated until the number of vectors in Γ_i reaches n. A test problem is thus specified by the triple (M, D, n).

These vector sets are then used to calculate $\bar{\Gamma}$ in Equation 2.55, which is the bottleneck of the exact value iteration algorithm. Monahan's Enumeration Al-

gorithm described in Section 2.6 will be used to create all possible vectors. Here, $M = |\Theta|$ is a substitute for the cardinality of the observation set. Increasing M means an exponential increase in the number of vectors, n^M.

Figure 3.4 shows the experimental results for 30 trials for $D = 5, 10, 15$. At each graph, the horizontal axis shows the initial number of dirty vectors, $|F_0| = N = 125, 625, 3125$ and the vertical axis demonstrates the time spent by the pruning algorithms. Both axes are given in logarithmic scale. The graphs show that FC algorithm has a time advantage in the order of magnitude compared to both the conventional algorithms and their revised versions. This is due to holding statistics about the slack variables and using this information to manipulate the number of vectors in the simplex tableau. By this idea, the number of vectors inside the simplex tableau stay in the order of D whereas for the other cases, the number of vectors increase at every iteration.

It can also be observed from the graphs that the performance of ISKY and ISwM algorithms become similar as the D increases. This is due to two facts. First, the advantage of checking multiple objective functions at every simplex iteration vanishes as the number of vertices increases. Second, as the vector indices are selected from a uniformly distributed random variable when $N = |F_0|$ is held constant, the number of dominated vectors decrease for the artificial examples with increasing D. To show this fact, mean value of the number of clean vectors $|Q|$ when $N = 5^5$ is also supplied in Figure 3.4 for $D = 5, 10, 15$. It can be seen that as $|Q|$ approaches to the value of N, the dimension of the simplex tableau becomes a bigger problem. As can be observed from Figure 3.4, LRwI is always better than the performance of LRK where the former uses a stratgy to select the objective function and the latter is doing this randomly.

3.5.2 Pruning Performance of Benchmark Problems

We then tested different pruning algorithms on a set of benchmark problems from [60]. As the proposed algorithms can be used for the general pruning procedure of any set of vectors, the total time spent in all the LPs in Equation 2.42 is given in Table 3.2. This total time is equal to the time spent in projection pruning

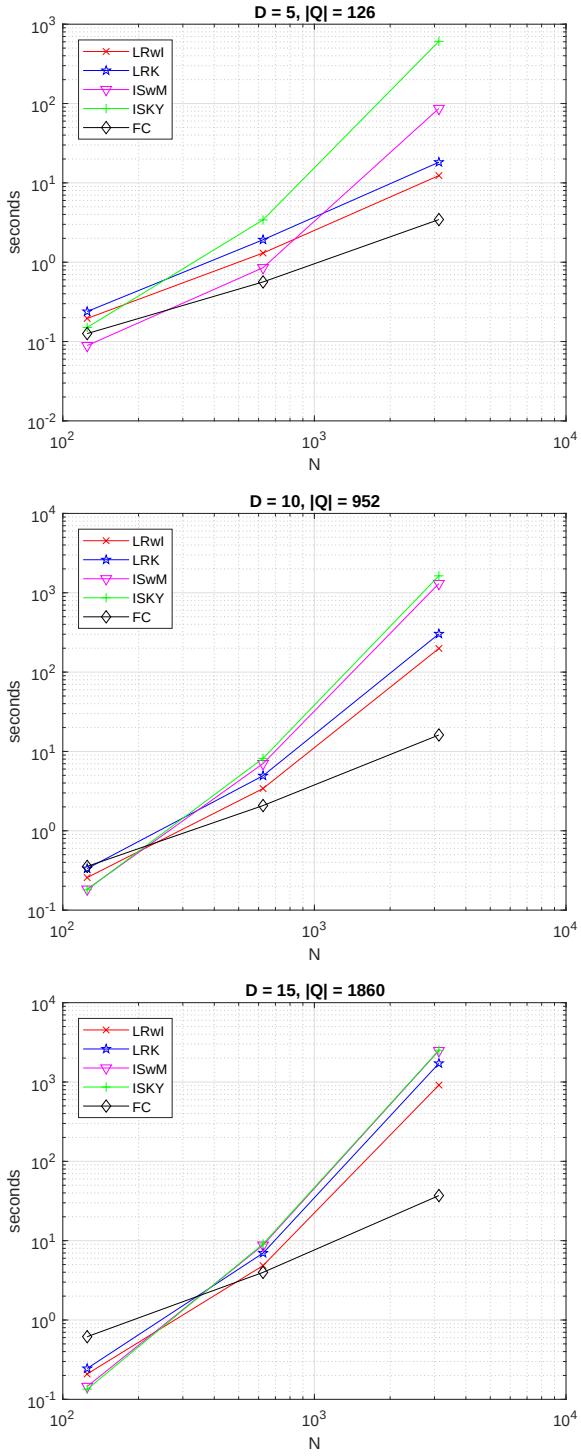


Figure 3.4: Mean time spent by different pruning algorithms (FC,ISKY,ISwM,LRK,LRwI) for artificial problems in 30 trials. An artificial problem is defined by the triple (M, D, n). Three cases where $D = 5, 10, 15$ are examined in different graphs. Horizontal axis shows the initial number of dirty vectors $N = n^M$ and the vertical axis demonstrates the time spent by the pruning algorithms. Each graph shows three different results for $n = 5$ and $M = 3, 4, 5$. $|Q|$ shows the average number of clean vectors for $N = 5^5$ after the pruning algorithms are terminated.

of action and observation dependent vectors in Equation 2.47, the pruning of the action dependent vectors after the cross sum in Equation 2.46 is performed by Monahan's enumeration algorithm, and the maximization given in Equation 2.45 among action dependent vectors.

The specific structure of Equation 2.55 has been exploited in [43]. Yet, our aim here is to show the effectiveness of FastCone algorithm for any given set of dirty vectors $\bar{\Gamma}$, without any prior information. For this aim, all possible vectors are first created as shown in Equation 2.55, and the pruning algorithms are applied to this set as in Equation 2.56.

Table 3.2 shows the performance results for different pruning algorithms for different benchmark problems. The table provides the specified time horizon for each problem, h, the number of clean vectors at the end of the specified horizon, $|Q|$, and the time spent in algorithms in milliseconds. At $h = 0$, it is assumed that there are no accumulated rewards for any of the states. Different time horizons are chosen regarding the complexity of the specific benchmark problem. As we are asked for the value function for a finite horizon, the discount factor is taken as 1 in any of the cases.

Table 3.2: Tests with benchmark problems in milliseconds

	Problem	h	$\|Q\|$	time	Problem	h	$\|Q\|$	time
LRwI	4x3.95	7	128	9443	hanks.95	9	17	2113.3
LRK			128	15566			17	3839.6
ISwM			129	558.1			17	73.8
ISKY			129	675.5			17	110
FC			128	**711.8**			17	**88.4**
LRwI	tiger.95	8	21	931.8	4x4.95	7	20	902
LRK			21	2008			20	2169
ISwM			21	28.5			20	68.3
ISKY			21	42.6			20	67.4
FC			21	**40.7**			20	**78**
LRwI	shuttle.95	6	185	12195	4x5x2.95	5	74	3875.6
LRK			185	13827			74	8347.7
ISwM			186	472.9			74	308.5
ISKY			186	550.6			74	263.7
FC			187	**658.9**			74	**778.1**

Table 3.2 shows that for benchmark problems, the FastCone algorithm has a similar performance to the Skyline based algorithms. This is due to two facts; most of the vectors in the dirty set are non-dominated vectors, therefore Lark based algorithms lose some time with writing the LPs from scratch. Secondly, as discussed in [51], Skyline algorithm has a comparable time advantage to Lark's algorithm when $|\bar{\Gamma}|$ is small. With increasing time horizon, $|\bar{\Gamma}|$ would increase and this time advantage would disappear. Therefore, it is possible to increase the time horizon of each problem to underline the performance of the FastCone algorithm as shown in Section 3.5.1.

Yet, with increasing time horizon, another problem arises. Recall that, even when the discount factor is taken as 1, the distribution of the vectors in the vector space is dependent on the properties of $\mathbf{R}(a, o)$ in Equation 2.53. With increasing horizon, the support set of some vectors become quite small, causing the LP to do many unnecessary simplex iterations. In our application, to assure the numerical stability of the pruning algorithms, two different tolerances were defined for the pivot and sign check operations.

For the pivoting operation, Harris ratio test is applied [47]. In the ratio test, the selection of the $\epsilon_{piv} = 10^{-8}$ value is critical especially for almost-degenerate cases where the support region of some of the vectors becomes very small. Simplex iterations also require a sign check operation to determine the pivoting column. To alleviate the roundoff error accumulation, every value smaller that $\epsilon_0 = 10^{-5}$ is assumed to be zero. However, due to the computation precision, small roundoff errors can accumulate and get over a predetermined threshold causing some extraneous vectors to appear in the clean set as shown in 3.2. Still, the number of clean vectors in any of the problems are almost identical which manifests the algorithm robustness. In the authors opinion, the importance of the benchmark problems is not to show the algorithm performance in time but in stability.

3.6 Conclusion

In the dynamic programming update of POMDPs, the number of vectors increase exponentially. Therefore, pruning this set of vectors to a minimal set becomes a major concern. Many scholars analyzed the steps of the exact value iteration algorithm to overcome the exponential increase in vectors [42–45]. Yet, there are only a few papers that specifically attack the linear programs which are being used by the exact value iteration algorithm [36,37,51]. As stated in [51], treating the LPs as a black box and focusing only on its optimal solution, results in a major performance loss. If the LP iterations are not taken into account, similar LP operations should be repeated for every dirty vector in $\bar{\Gamma}$.

In our former study [36], we have discussed that it is possible to define a heuristic 'closeness' measure for the dirty vectors. This idea is described in Section A.2 and used to improve the Lark's Algorithm. To be able to make a formal definition of this closeness measure, this study puts forward a geometric approach for the vector pruning problem. The use of convexity to define the vector pruning problem and the geometry of convex regions described in Section 3.3 is a novel idea. This is the first contribution of this chapter.

The second contribution is the algebraic approach that is given in tandem with the geometric approach in Section 3.3. For the geometric approach, the problem is described by small sized LPs, called $LP_{A,0}$ that refer to any selection of active set of vectors Γ_A and a dirty vector $\boldsymbol{\gamma_0}$. The simplex iterations given in Algorithm 4, LPINIT1 and LPINIT2 applied to $LP_{A,0}$ are also explained geometrically by the theorems given in Section 3.3.

As stated in Section 3.4.1, any of the pruning algorithms is basically the selection of a clean set Γ_A for a given dirty vector $\boldsymbol{\gamma_0}$. The final contribution of this study is the use of stated framework for a novel algorithm called FastCone (FC). FC algorithm is compared to the conventional algorithms and their revised versions both analytically and experimentally. It is shown that FC algorithm results in an order of magnitude performance increase as the problem size increases.

One recent study has also shown how the number of constraints in the LP

can be reduced by using the Bender's decomposition technique [37]. FastCone algorithm also uses the dual problem for finding the solution of the LP but is different from [37] in three senses. First, it checks if the elimination criteria for the selected dirty vector is satisfied at every simplex iteration and terminates the optimization if possible. This is important, as it might be possible to eliminate a vector without reaching to the optimal solution of the linear program. After termination, the same simplex tableau is used for the elimination of another vector. This is the second difference. Thirdly, this new vector is selected by a defined closeness metric that allows the solutions to be close to each other, meaning a smaller number of LP iterations.

While FastCone is an efficient algorithm as shown in part 3.5, vector pruning is an open problem because of its different structure. In the vector pruning problem there are as many objective functions as the number of linear constraints. Therefore the order, in which these constraints are selected as the objective function, results in a major performance difference. In our future work, we aim to exploit the framework stated in this thesis for proposing both exact and approximate algorithms.

CHAPTER 4

EXTENDING THE THEORETICAL FRAMEWORK FOR THE CROSS-SUM OPERATION

4.1 Introduction

In Chapter 2, we have discussed the exact value iteration algorithm. Section 2.6 introduced the enumeration algorithm for exact value iteration which is an upper bound for the computational complexity of the problem. Here, all possible vectors are first created and then pruned to a minimal set of non-dominated vectors. This pruning operation is explained in Chapter 3.

The computational complexity of the exact value iteration algorithm is mainly due to the cross-sum operation defined over a set of vectors shown in Equation 2.55. The number of vectors created at the cross-sum step are $|\bar{\Gamma}_t^a| = \prod_{o\in\Theta} |\Gamma_t^{a,o}| \approx |\Gamma_{t+1}|^{|\Theta|}$. However, the creation of the set $\bar{\Gamma}_t$ is not necessarily as costly as discussed in Section 2.6. There are many algorithms that use different properties of the cross-sum operation which is a summation over different PWLC functions. In this chapter, we will first introduce these algorithms and then use the framework proposed in Chapter 3 to offer a novel algorithm.

4.1.1 Conventions

Denote $M := |\Theta|$ as the observation cardinality. We have M sets $\{\Gamma_m\}_{m=1}^M$, where each Γ_m denotes a set of vectors. It is reasonable to think that Γ_m is a set of clean vectors as it is possible to prune this set before starting the cross-

sum operation. That is, $\Gamma_m = \mathbb{PR}(\Gamma_m)$. Let $N_m := |\Gamma_m|$. Then we can write $\Gamma_m = \{\boldsymbol{\gamma}_{m,1}, \ldots, \boldsymbol{\gamma}_{m,N_m}\}$. Each set Γ_m can also be described by the set of indices $Q_m = \{1, \ldots, N_m\}$.

We will define a recursive structure for the cross-sum operation. Define $\bar{\Gamma}_{k:M} := \bigoplus_{m=k}^{M} \Gamma_m$, $\Gamma_{k:M} := \mathbb{PR}(\bar{\Gamma}_{k:M})$ and $N_{k:M} := |\Gamma_{k:M}|$. For the indices, define $Q_{k:M} = Q_k \times Q_{k+1} \times \ldots \times Q_M$. The vectors in $\bar{\Gamma}_{k:M}$ can be uniquely defined by the indices in $Q_{k:M}$.

Γ denotes the clean set after the cross-sum operation. That is $\Gamma := \Gamma_{1:M}$. Denote $N := N_{1:M}$. Then we can write $\Gamma = \{\boldsymbol{\gamma}_1, \ldots, \boldsymbol{\gamma}_N\}$. Recall that any element $\boldsymbol{\gamma}_i \in \bar{\Gamma}_{1:M}$ can be represented by $(i_1, i_2, \ldots, i_M) \in Q_{1:M}$ where i_j corresponds to the i_jth element in set Γ_j. Then $(i_1, i_2, \ldots, i_M) \in Q_{1:M} \iff \boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$.

The support set of vector $\boldsymbol{\gamma}_{m,k}$ in Γ_i will be denoted by $\mathcal{B}_{m,k} := R(\boldsymbol{\gamma}_{m,k}, \Gamma_m)$. The support set of vector $\boldsymbol{\gamma}_j$ in Γ will be denoted by $\mathcal{B}_j := R(\boldsymbol{\gamma}_j, \Gamma_{1:M})$.

The dirty set of vectors, $\bar{\Gamma}$, will also be denoted by the set $F_0 = \{1, \ldots, N\}$, where the vectors will be represented by their indexes. As we give our decision about the vectors in the dirty set $\bar{\Gamma}$, the size of F_0 decreases. For notational convention, we will define F, which represents the index of current dirty set of vectors, respectively. At initialization, the dirty set contains all the vectors, $F = F_0$. All algorithms continue until $F = \emptyset$. In a similar manner, we will define Q_∞ and Q which describe the final and current clean set of vectors, respectively. The indexes of the vectors in Γ are one-to-one correspondent with the set Q_∞; that is, $\Gamma = \bar{\Gamma}_{Q_\infty}$. At the beginning of the algorithm, $Q = \emptyset$ and when the algorithm is terminated, $Q = Q_\infty$.

4.2 Known Exact Value Iteration Algorithms

Smallwood et al. noted that the cross-sum operation partitions the belief set in a particular way [20]. The special structure of the cross-sum operation can be summarized by the following Lemma 4.2.1:

Lemma 4.2.1. *Suppose we have* $\{\Gamma_m\}_{m=1}^M$. *Take* $\boldsymbol{\gamma}_i = \sum_{m=1}^M \boldsymbol{\gamma}_{m,i_m}$. *Then,*

$$\mathcal{B}_i = \bigcap_{m=1}^{M} \mathcal{B}_{m,i_m} \tag{4.1}$$

Proof. ($\Rightarrow$:) Take $b \in \mathcal{B}_i$ and assume that $\exists l$ such that $b \notin \mathcal{B}_{l,i_l}$. Then $\exists k$ such that $b \in \mathcal{B}_{l,k}$. Form $\tilde{\boldsymbol{\gamma}}_i = \sum_{m \neq l} \boldsymbol{\gamma}_{m,i_m} + \boldsymbol{\gamma}_{l,k}$. Note that $b\tilde{\boldsymbol{\gamma}}_i > b\boldsymbol{\gamma}_i$. As $\tilde{\boldsymbol{\gamma}}_i \in \bar{\Gamma}_{1:M}$, $\exists \boldsymbol{\gamma}_j \in \Gamma_{1:M}$ such that $b\boldsymbol{\gamma}_j \geq b\tilde{\boldsymbol{\gamma}}_i$. Then $b\boldsymbol{\gamma}_j > b\boldsymbol{\gamma}_i$. This is not possible as $b \in \mathcal{B}_i$. Therefore $b \in \mathcal{B}_{l,i_l}$.

($\Leftarrow$:) Take $\boldsymbol{\gamma}_i = \sum_m \boldsymbol{\gamma}_{m,i_m}$ and $b \in \bigcap_{m=1}^M \mathcal{B}_{m,i_m}$. Assume $\exists \boldsymbol{\gamma}_j \in \Gamma_{1:M}$ such that $b\boldsymbol{\gamma}_j > b\boldsymbol{\gamma}_i$. We know that $\boldsymbol{\gamma}_j = \sum_m \boldsymbol{\gamma}_{m,j_m}$. Then $\exists m$ such that $b\boldsymbol{\gamma}_{m,j_m} > b\boldsymbol{\gamma}_{m,i_m}$. Contradiction. ■

There are many algorithms that exploit the special structure in the cross-sum operation of many sets of vectors to reduce the number of constraints in the LPs. Before going into the algorithms, we want to introduce some mathematical definitions that would help us with explaining these algorithms.

Definition 15. *For a given set of vector sets* $\{\Gamma_m\}_{m=1}^M$, *define the following operator* $\mathbb{I}$*;*

$$\mathbb{I}(\Gamma_1, \ldots, \Gamma_M) = \left\{ (i_1, \ldots, i_M) | \bigcap_m \mathcal{B}_{m,i_m} \neq \emptyset \text{ where } i_m \in Q_m,\ 1 \leq m \leq M \right\}$$

When a belief subset $\tilde{\mathcal{B}}$ *is also given, the operator becomes;*

$$\mathbb{I}(\tilde{\mathcal{B}}; \Gamma_1, \ldots, \Gamma_M) = \left\{ (i_1, \ldots, i_M) | \left(\bigcap_m \mathcal{B}_{m,i_m} \right) \bigcap \tilde{\mathcal{B}} \neq \emptyset \text{ where } i_m \in Q_m,\ 1 \leq m \leq M \right\}$$

The following equation shows the relation between the set and index notations.

$$\sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m} \in \Gamma \iff (i_1, \ldots, i_M) \in \mathbb{I}(\Gamma_1, \ldots, \Gamma_M)$$

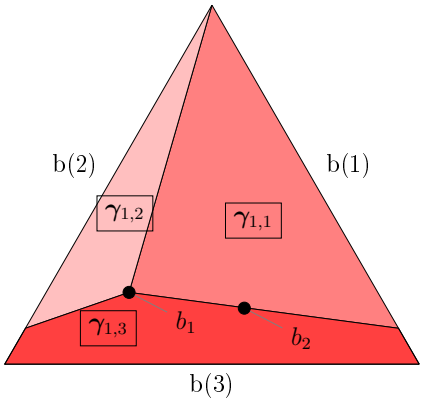
Therefore $|\Gamma| = |\mathbb{I}(\Gamma_1, \ldots, \Gamma_M)|$. Operator given in Definition 15 is defined by Feng et al. [61]. The following lemma shows clearly the iterative logic of the cross-sum operation.

Lemma 4.2.2. *Given* $\{\Gamma_m\}_{m=1}^M$, *the following statements are true.*

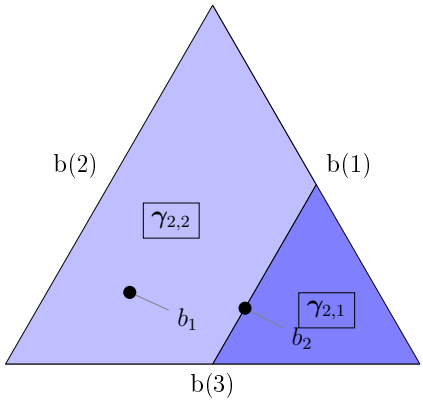
$$(i_k, \ldots, i_M) \in \mathbb{I}(\Gamma_k, \ldots, \Gamma_M) \Rightarrow (i_{k+1}, \ldots, i_M) \in \mathbb{I}(\Gamma_{k+1}, \ldots, \Gamma_M)$$

$$(i_k, \ldots, i_M) \in \mathbb{I}(\tilde{\mathcal{B}}; \Gamma_k, \ldots, \Gamma_M) \Rightarrow (i_{k+1}, \ldots, i_M) \in \mathbb{I}(\tilde{\mathcal{B}}; \Gamma_{k+1}, \ldots, \Gamma_M)$$

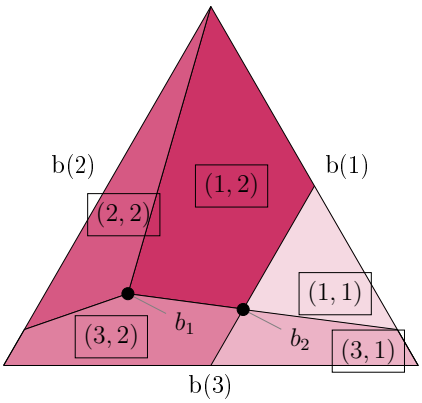
Proof. Take $(i_k, \ldots, i_M)$. If $\bigcap_{m=k+1}^M \mathcal{B}_{m,i_m} = \emptyset$, then $\bigcap_{m=k}^M \mathcal{B}_{m,i_m} = \emptyset$. Therefore, $(i_k, \ldots, i_M) \notin \mathbb{I}(\Gamma_k, \ldots, \Gamma_M)$. ■



(a) Belief State Partition of set Γ_1



(b) Belief State Partition of set Γ_2



(c) Belief State Partition of set $\Gamma_1 \oplus \Gamma_2$

Figure 4.1: Cross-sum of two sets $D = 3$

The belief state representation of the value function can be seen in Figure 4.1. In the figure, a simple case of the cross-sum operation where $D = 3$ and $M = 2$ is investigated. The support set of the vectors in the figure are denoted by the vector names in Figures 4.1a and 4.1b. In Figure 4.1c, the support set of the vectors is shown by their index numbers (i_1, i_2). As can be seen from the figure, for the belief point b_1, there are D active vectors from the set Γ_1. Therefore, point b_1 is a vertex point of set Γ_1. If a belief point b_1 is a vertex of one of

the input sets, it naturally becomes a vertex of the output set. Discarding degeneracies, the point b would be in the support set of a single vector for the other input sets of the cross-sum operation (in our case $\boldsymbol{\gamma}_{2,2}$). One important observation can be made here. If there are less than D vectors for any of the input sets Γ_i, this results in a degeneracy. The solution at belief point b_2 is such a case. There are two active vectors from both Γ_1 and Γ_2 at point b_2 and the cross-sum of all vectors are inside the clean set. Lemma 4.2.1 says that $\boldsymbol{\gamma}_{1,2}+\boldsymbol{\gamma}_{2,1}$ cannot be an active vector as $\mathcal{B}_{1,2} \cap \mathcal{B}_{2,1} = \emptyset$. But finding if the regions of two vectors from different sets is possible only by writing the suitable LPs.

It is also possible to investigate the cross-sum operation in the vector space representation of the value function. In [38], Zhang uses the vector space formulation particularly for the cross-sum operation of different sets of vectors. It is known that optimal value function in the vector space forms a convex polyhedron and the cross-sum operation is the Minkowski addition of the the vectors that are forming these convex polyhedrons. If the vertices of each convex polyhedron has already been enumerated [52], this enumeration information can be used when writing the linear programs for finding the vertices of the resultant convex polyhedron. For the pruning operation, the aim is not to enumerate all the vertices of the value function [52, 62], but to travel the least number of vertices until we can assure that all the vectors in our the final set are non-dominated.

4.2.1 Incremental Pruning Algorithm

First systematic approach to deal with the pruning operation for the cross-sum operation is offered by Cassandra et al. [43] with the incremental pruning algorithm. Incremental pruning exploits the fact that $\mathbb{PR}$ and $\bigoplus$ operators in Equations 2.55 and 2.56 can be interleaved. In mathematical terms,

$$\mathbb{PR}(\Gamma_1 \oplus \Gamma_2 \oplus \Gamma_3) = \mathbb{PR}(\Gamma_1 \oplus \mathbb{PR}(\Gamma_2 \oplus \Gamma_3)) \tag{4.2}$$

This idea is useful while $|\mathbb{PR}(\Gamma_2 \oplus \Gamma_3)|$ can be much less compared to $|\Gamma_2 \oplus \Gamma_3| = N_2 \times N_3$. Therefore, if the pruning operation is done before cross-sum operation with Γ_1, the number of vectors created will be less compared to operand of the

left hand-side of Equation 4.2. In a similar manner when $M > 3$,

$$\mathbb{PR}\left(\bigoplus_{m=1}^{M}\Gamma_m\right) = \mathbb{PR}\left(\Gamma_1 \oplus \mathbb{PR}\left(\Gamma_2 \oplus \ldots \mathbb{PR}\left(\Gamma_{M-1} \oplus \Gamma_M\right)\ldots\right)\right) \tag{4.3}$$

The order of the sets in Equation 4.3 does not change the resultant clean set Γ. Cassandra et al. [43] also observed that, it might be beneficial to choose the sets Γ_i and Γ_j in the inner loops of Equation 4.3 if $|\mathbb{PR}(\Gamma_i \oplus \Gamma_j)| << |\Gamma_i \oplus \Gamma_j|$.

As can be seen Equation 4.3 has a recursive structure. To find $\Gamma_{k:M}$, first $\Gamma_k \oplus \Gamma_{k+1:M}$ is calculated and $\mathbb{PR}(\Gamma_k \oplus \Gamma_{k+1:M}) = \Gamma_{k:M}$ is performed. In Chapter 3, we have seen two different pruning operators; Lark's and iterative skyline algorithms. Both algorithms write an LP for any of these vectors inside $\bar{\Gamma}_{k:M}$ to see if the vector is dominated. We need to solve $N_k \times N_{k+1:M}$ LPs to reach the set $\Gamma_{k:M}$. For the Lark's algorithm, the number constraints inside the LP increases from 1 to $N_k \times N_{k+1:M} + 1$ as the algorithm proceeds. If we have selected the Skyline Algorithm, the constraints in each LP would be $N_k \times N_{k+1:M} + 1$. Note that this operation should be repeated for k going from N to 1. Note that at every step, we have two set of vectors that are being cross-summed.

4.2.2 Generalized Incremental Pruning

Cassandra et. al. gives an elaborate way to use Lemma 4.2.1 in the cross-sum step of exact value iteration algorithm. The generalized incremental pruning algorithm is described in Algorithm 5. The algorithm is basically a recursive call of the subroutine GCS which is a generalized version of the cross-sum of two set of vectors. Any element in the dirty set F is represented by the index set (i_1, i_2). The algorithm has a similar structure to Lark's algorithm given in Algorithm 1. Two routines are taken from Algorithm 1; FNDBLF and BEST. The FNDBLF routine in Step 15 is used to find a belief point for a vector with respect to a given set of vectors. BEST routine in Step 19 is used to select one of the dominating vectors if a belief state is given. This algorithm is used to find a witness point for a vector $\boldsymbol{\gamma}_i$ with respect to the set Ψ. If the vector has a witness point with respect to the set Ψ, then it is a non-dominated vector. Ψ can be one of the following sets:

Algorithm 5 Generalized Incremental Pruning

procedure GIP($\Gamma_1, \ldots, \Gamma_M$)
  $\Gamma \leftarrow \Gamma_M$
  $k \leftarrow M - 1$
  while $k \neq 0$ **do**
    $\Gamma \leftarrow$ GCS (Γ, Γ_k)
    $k \leftarrow k - 1$
  end while
  return Γ
end procedure
procedure GCS(Γ_1, Γ_2)
  $\bar{\Gamma} \leftarrow \bar{\Gamma}_{1:2}$, $F \leftarrow Q_1 \times Q_2$, $Q \leftarrow \emptyset$
  while $F \neq \emptyset$ **do**
    $(i_1, i_2) \leftarrow$ any element in F
    $\boldsymbol{\gamma}_i \leftarrow \boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,i_2}$
    $(\delta, b) \leftarrow$ FNDBLF($\boldsymbol{\gamma}_i, \Psi$)
    if $\delta > 0$ **then**
      $F \leftarrow F \setminus \{(i_1, i_2)\}$
    else
      $\boldsymbol{\gamma} \leftarrow$ BEST($b, \bar{\Gamma}$)
      $(i_1, i_2) \leftarrow$ index of $\boldsymbol{\gamma}_i$ in $\bar{\Gamma}$
      $F \leftarrow F \setminus \{(i_1, i_2)\}$
      $Q \leftarrow Q \cup \{(i_1, i_2)\}$
    end if
  end while
  return $\bar{\Gamma}_Q$
end procedure

1. $\Psi = \Gamma_1 \oplus \Gamma_2$
2. $\Psi = \bar{\Gamma}_Q$
3. $\Psi = (\{\boldsymbol{\gamma}_{1,i_1}\} \oplus \Gamma_2) \cup (\Gamma_1 \oplus \{\boldsymbol{\gamma}_{2,i_2}\})$
4. $\Psi = (\{\boldsymbol{\gamma}_{1,i_1}\} \oplus \Gamma_2) \cup \Psi_2$
5. $\Psi = \Psi_1 \cup (\Gamma_1 \oplus \{\boldsymbol{\gamma}_{2,i_2}\})$

where $\Psi_1 = \{\boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,k} | (i_1, k) \in Q\}$ and $\Psi_2 = \{\boldsymbol{\gamma}_{1,k} + \boldsymbol{\gamma}_{2,i_2} | (k, i_2) \in Q\}$. Recall that before running the FNDBLF routine, we select a vector (i_1, i_2). For the details of the algorithm, the reader can refer to [44]. We want to show what the selection of different Ψ sets means.

When $\Psi = \Gamma_1 \oplus \Gamma_2$ is selected, all the vectors are written inside the LP and the algorithm becomes similar to the LP of the Skyline algorithm given in Algorithm 3. When $\Psi = \bar{\Gamma}_Q$ is selected, the algorithm becomes similar to the Lark's algorithm as only the vectors in the clean set are considered inside the LP. For the other cases, we want to keep in mind that FNDBLF routine writes

$$b(\boldsymbol{\gamma}_i - \hat{\boldsymbol{\gamma}}) + \delta > 0 \quad \forall \hat{\boldsymbol{\gamma}} \in \Psi$$

Recall that $\boldsymbol{\gamma}_i = \boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,i_2}$.

When $\Psi = (\{\boldsymbol{\gamma}_{1,i_1}\} \oplus \Gamma_2) \cup (\Gamma_1 \oplus \{\boldsymbol{\gamma}_{2,i_2}\})$ is selected, the constraints inside the LP becomes

$$b(\boldsymbol{\gamma}_{m,i_m} - \boldsymbol{\gamma}_{m,t_m}) + \delta > 0, \quad \forall t_m \in Q_m \setminus \{i_m\},\ 1 \leq m \leq 2 \tag{4.4}$$

where for any $\delta \leq 0$, we have found a $b \in \mathcal{B}_{1,i_1} \cap \mathcal{B}_{2,i_2}$.

For the 4th and 5th cases there is a symmetry. For this, it is enough to investigate only one of these cases. When $\Psi = (\{\boldsymbol{\gamma}_{1,i_1}\} \oplus \Gamma_2) \cup \Psi_2$ is selected,

$$\begin{aligned} b(\boldsymbol{\gamma}_{1,i_1} - \boldsymbol{\gamma}_{1,t_1}) + \delta > 0, &\quad \forall t_1 | \boldsymbol{\gamma}_{1,t_1} + \boldsymbol{\gamma}_{2,i_2} \in \Psi_2 \\ b(\boldsymbol{\gamma}_{2,i_2} - \boldsymbol{\gamma}_{2,t_2}) + \delta > 0, &\quad \forall t_2 \in Q_2,\ t_2 \neq i_2 \end{aligned}$$

where the second set of constraints restricts the region and the first set does the pruning operation over this region. This algorithm in great similarity with

operating the Lark's algorithm over the belief region $\mathcal{B}_{2,i_2}$. This variant of the incremental pruning algorithm has the smallest number of constraints in the LP. Yet, even in this case, the algorithm is defined only for two sets, therefore the effectiveness of the algorithm diminishes as more sets are cross-summed. For instance, when $M = 3$, after the sets $\Gamma_2 \oplus \Gamma_3$ is cross-summed, the number of constraints inside the LP would vary between $|\Gamma_{2:3}|$ to $N_1 + |\Gamma_{2:3}|$. With a similar reasoning, when we consider Equation 4.3 where $M > 3$, the number of constraints inside the LP will approach $|\Gamma_{1:M}|$.

As we have discussed before, Lemma 4.2.1 is first noticed by Smallwood et al. [20] and later developed by Cassandra [43]. Feng et al. [45] have built the restricted region variants of the incremental pruning algortihm on this background. Therefore, the algorithms proposed by Feng et al. [61] are a major improvement to the generalized incremental pruning algorithm when $M > 2$. The generalization of 3rd case will be discussed in detail in the region intersection algorithm given in Algorithm 6 and 4th and 5th cases in restricted region algorithm given in Algorithm 7, respectively.

4.2.3 Intersection Based Incremental Pruning

Feng et al. [61] generalize the proposition of Lemma 4.2.1 as follows: Suppose we want to determine if a particular vector $\boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$ is dominated. Instead of calculating the vector set $\bar{\Gamma}_{1:M}$ and then searching for $\boldsymbol{\gamma}_i$, we can write an LP using the intersection of the support set of individual vectors $\boldsymbol{\gamma}_{m,i_m}$. This is equivalent to the following LP;

$$\begin{aligned} &\min 0 \\ &b(\boldsymbol{\gamma}_{m,i_m} - \boldsymbol{\gamma}_{m,t}) > 0, \quad \forall t \in Q_m \setminus \{i_m\},\ 1 \leq m \leq M \\ &\sum_{l=1}^{D} b(l) = 1 \\ &b(l) \geq 0, \quad 1 \leq l \leq D \end{aligned} \tag{4.5}$$

However, the LP in Equation 4.5 should be solved for any $(i_1, \ldots, i_M)$ which means $\prod_i N_i$ LPs where every LP has $\sum_m N_m + 1$ constraints. Instead of writing an LP for every possible vector, Feng et al. [61] proposes to use Lemma 4.2.2 and

stop the recursion for any branch at step k if $(i_{k+1}, \dots, i_M) \in \mathbb{I}(\Gamma_{k+1}, \dots, \Gamma_M)$ but $(i_k, \dots, i_M) \notin \mathbb{I}(\Gamma_k, \dots, \Gamma_M)$.

The algorithm for finding $\mathbb{I}(\Gamma_1, \dots, \Gamma_M)$ via region intersection is given in Algorithm 6. The main algorithm, IBIP, is a recursive call of the function PRI, where at each time step the set $\Gamma_{k:M} = \mathbb{PR}(\Gamma_k \bigoplus \Gamma_{k+1:M})$ is found. The function PRI writes an LP for every vector in $(i_k, \dots, i_M) \in Q_k \times \mathbb{I}(\Gamma_{k+1}, \dots, \Gamma_M)$. Procedure LPINTSCT corresponds to the LP for region intersection described in Equation 4.5. The function returns true if the regions of the given vectors in the n-tuple intersect. The algorithm selects $(i_k, \dots, i_M)$ randomly from set F and tries to find if their support set is intersecting. Note that as there is no objective function, any point $b \in \bigcap_{j=k}^{M} \mathcal{B}_{m,i_m}$ satisfies the linear program. If we start the LP from $b \notin \bigcap_{m=k}^{M} \mathcal{B}_{m,i_m}$, the program finds a corner point of the region $\bigcap_{m=k}^{M} \mathcal{B}_{m,i_m}$. If $\bigcap_{m=k}^{M} \mathcal{B}_{m,i_m} = \emptyset$ then $(i_k, \dots, i_M) \notin \mathbb{I}(\Gamma_k, \dots, \Gamma_M)$.

We can see that there are $N_k \times |\mathbb{I}(\Gamma_{k+1}, \dots, \Gamma_M)| = N_k \times N_{k+1:M}$ LPs needed which is the same for the incremental pruning algorithm. However for each LP, there are approximately $\sum_{m=k}^{M} N_m$ constraints compared to $N_k \times N_{k+1:M}$ constraints in the incremental pruning algorithm. Note that the LP structure is somehow similar to the Skyline algorithm where all the vectors in $\bigcup_{m=k}^{M} \Gamma_m$ are written inside the LP as constraints. Moreover, it is possible to decrease the number of vectors inside the LP by using a similar logic to Lark's algorithm. This will be investigated by the following algorithm.

4.2.4 Region Based Incremental Pruning

Feng et al. [61] also propose a method to decrease the number of constraints in each LP to be solved. Recall that Lark's algorithm in Section 3.2.1 tries to find a belief point $b \in \mathcal{B}$ at the end of every LP which is used to either delete a vector from the dirty set or add a new vector to the clean set. Instead of the whole belief set, it is possible to define a subset of the belief set $\tilde{\mathcal{B}}$ and perform a similar operation. This is equivalent to a special case of the operator in Definition 15, where a single set Γ_i is given and $\mathbb{I}(\tilde{\mathcal{B}}; \Gamma_i)$ is performed.

Algorithm 6 Intersection Based Incremental Pruning

procedure IBIP$(\Gamma_1, \ldots, \Gamma_M)$
  $\bar{\Gamma} \leftarrow \bar{\Gamma}_{1:M}$
  $k \leftarrow M$
  $Q \leftarrow Q_k$
  while $k \neq 0$ **do**
    $Q \leftarrow$ PRI $(\{\Gamma_m\}_{m=k}^{M}, Q)$
    $k \leftarrow k - 1$
  end while
  return Q
end procedure
procedure PRI$(\{\Gamma_m\}_{m=k}^{M}, \bar{Q})$
  $Q \leftarrow \emptyset$
  $F \leftarrow Q_k \times \bar{Q}$
  while $F \neq \emptyset$ **do**
    $(i_k, \ldots, i_M) \leftarrow$ any element in F
    $F \leftarrow F \setminus \{(i_k, \ldots, i_M)\}$
    if LPINTSCT $((i_k, \ldots, i_M), \{\Gamma_m\}_{m=k}^{M}) ==$ true **then**
      $Q \leftarrow Q \cup \{(i_k, \ldots, i_M)\}$
    end if
  end while
  return $\bar{\Gamma}_Q$
end procedure
procedure LPINTSCT$((i_k, \ldots, i_M), \{\Gamma_m\}_{m=k}^{M})$
  solve the following linear program
  $\min 0$ subject to
  $b(\boldsymbol{\gamma}_{m,i_m} - \boldsymbol{\gamma}_{m,t_m}) > 0, \quad \forall t_m \in Q_m \setminus \{i_m\},\ k \leq m \leq M$
  $\sum b(l) = 1,\ 1 \leq l \leq D$
  $b \geq \mathbf{0}$
  if LP is feasible **return** true
  else **return** false
end procedure

Assume that $M = 2$ and $\Gamma = \mathbb{PR}(\Gamma_1 \oplus \Gamma_2)$. The question is for a fixed $\boldsymbol{\gamma}_{2,i_2} \in \Gamma_2$ which vectors from $\boldsymbol{\gamma}_{1,i_1} \in \Gamma_1$ should be selected such that $\boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,i_2} \in \Gamma$. As discussed in [45], the benefit of the region-based pruning algorithm comes from the following fact;

$$\mathbb{I}(\Gamma_1, \Gamma_2) = \bigcup_{i_2 \in Q_2} \{\mathbb{I}(\mathcal{B}_{2,i_2}; \Gamma_1) \times \{i_2\}\} \tag{4.6}$$

Note that, in this case the clean set of vectors is only the set of vectors in Γ_1 which are dominant over $\mathcal{B}_{2,i_2}$. This is equivalent to saying $i_1 \in \mathbb{I}(\mathcal{B}_{2,i_2}; \Gamma_1)$. Therefore, the vectors in Γ_1 can be regarded as the dirty set of vectors. For the initialization of the algorithm $F = Q_1$ and $Q = \emptyset$. The algorithm picks a vector $\boldsymbol{\gamma}_{1,k} \in \Gamma_1$ and tries to find a belief point $b \in \mathcal{B}_{2,i_2}$ that satisfies $b\boldsymbol{\gamma}_{1,k} > b\boldsymbol{\gamma}_{1,l}$, $\forall l \in Q$. This is equivalent to the following LP;

$$\begin{aligned}
&\min \delta \\
&b(\boldsymbol{\gamma}_{1,k} - \boldsymbol{\gamma}_{1,t_1}) + \delta > 0, \quad \forall t_1 \in Q \subset Q_1 \\
&b(\boldsymbol{\gamma}_{2,i_2} - \boldsymbol{\gamma}_{2,t_2}) > 0, \quad \forall t_2 \in Q_2,\ t_2 \neq i_2 \\
&\sum_{l=1}^{D} b(l) = 1 \\
&b(l) \geq 0, \quad 1 \leq l \leq D
\end{aligned} \tag{4.7}$$

The optimal solution occurs at the belief state b_0 and the value of the objective function is δ_0. We know that $b_0 \in \mathcal{B}_{2,i_2}$. If δ_0 is less than 0, it means that there is a vector constraint from set Γ_2 that gives a higher value for y_2 at the belief state b_0 where the optimal solution occurs. The vector index $k' = \arg\max_{k \in F} b_0 \boldsymbol{\gamma}_{1,k}$ is added to the clean set Q and deleted from F. If δ_0 is greater than or equal to zero, the vector $\boldsymbol{\gamma}_{1,k}$ is dominated by the vectors in the clean set $\Gamma_{1,Q}$ and therefore k is deleted from F. The procedure continues until there are no vectors left in F.

Algorithm 7 is the restricted region variant of incremental pruning. The main routine is $\mathbb{I}(\mathcal{B}_{2,i_2}, \Gamma_1)$ where the pruning operation is performed only over the belief subset $\mathcal{B}_{2,i_2}$. To cover the whole belief set, we need to perform this operation $\forall i_2 \in Q_2$. Note that the algorithm can be written for any closed belief subset $\tilde{\mathcal{B}}$, the only constraint is that the borders of $\tilde{\mathcal{B}}$, should be determined by

the linear equations as demonstrated in Equation 4.7.

If we were to find $\mathbb{I}(\Gamma_1, \Gamma_2)$ by using Algorithm 6, the number of LPs to be solved are $N_1 \times N_2$. Same number of LPs are needed in Algorithm 7. However for the former algorithm, each LP would have $N_1 + N_2 + 1$ constraints. The number of constraints inside the LP for the latter varies between $N_1 + 1$ and $N_1 + N_2 + 1$.

Algorithm 7 Restricted Region Algorithm

procedure $\mathbb{I}(\mathcal{B}_{2,i_2}; \Gamma_1)$
  $Q \leftarrow \emptyset$
  $F \leftarrow Q_1$
  while $F \neq \emptyset$ **do**
    $i \leftarrow$ any element in F
    $(\delta, b) \leftarrow \text{LPRR}(\mathcal{B}_{2,i_2}, \boldsymbol{\gamma}_{1,k}, \Gamma_{1,Q})$
    if $\delta > 0$ **then**
      $\boldsymbol{\gamma} \leftarrow \text{BEST}(b, \Gamma_1)$
      $k \leftarrow$ index of $\boldsymbol{\gamma}$ in Γ_1
      $Q \leftarrow Q \cup \{k\}$
      $F \leftarrow F \setminus \{k\}$
    else
      $F \leftarrow F \setminus \{i\}$
    end if
  end while
  return $\Gamma_{1,Q}$
end procedure
procedure $\text{LPRR}(\mathcal{B}_{2,i_2}, \boldsymbol{\gamma}_{1,k}, \Gamma_{1,Q})$
  solve the following linear program
  variables: δ, b
  $\min \delta$ subject to
  $b(\boldsymbol{\gamma}_{1,k} - \boldsymbol{\gamma}_{1,t_1}) + \delta > 0, \quad \forall t_1 \in Q$
  $b(\boldsymbol{\gamma}_{2,i_2} - \boldsymbol{\gamma}_{2,t_2}) > 0, \quad \forall t_2 \in Q_2,\ t_2 \neq i_2$
  $\sum b(l) = 1$
  $b \geq \mathbf{0}$
  return (δ, b)
end procedure

What happens when $M > 2$? Feng et al. describe a recursive structure by using the following fact.

$$\mathbb{I}(\Gamma_1, \ldots, \Gamma_M) = \bigcup_{i_M} \{\mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_1, \ldots, \Gamma_{M-1}) \times i_M\} \tag{4.8}$$

$$= \bigcup_{(i_k, \ldots, i_M)} \left\{ \mathbb{I}\left(\bigcap_{m=k}^{M} \mathcal{B}_{m,i_m}; \Gamma_1, \ldots, \Gamma_{k-1} \right) \times (i_k, \ldots, i_M) \right\},\ 1 \leq k \leq M \tag{4.9}$$

Algorithm 8 Region Based Incremental Pruning

procedure RBIP($\Gamma_1, \ldots, \Gamma_M$)
  $\bar{\Gamma} \leftarrow \bar{\Gamma}_{1:M}$
  $F \leftarrow Q_M$
  $Q \leftarrow \emptyset$
  while $F \neq \emptyset$ **do**
    $i \leftarrow$ any element in F
    $F \leftarrow F \setminus \{i\}$
    $Q \leftarrow Q \cup \mathbb{I}(\mathcal{B}_{M,i}; \Gamma_1, \ldots, \Gamma_{M-1}) \times \{i\}$
  end while
  return $\bar{\Gamma}_Q$
end procedure
procedure $\mathbb{I}(\tilde{\mathcal{B}}; \Gamma_1, \ldots, \Gamma_K)$
  $Q \leftarrow \emptyset$
  $k \leftarrow K$
  while $k \neq 0$ **do**
    if $k = 1$ **then**
      $Q \leftarrow \mathbb{I}(\tilde{\mathcal{B}}; \Gamma_1)$
    else
      $F \leftarrow Q_k$
      while $F \neq \emptyset$ **do**
        $i \leftarrow$ any element in F
        for $1 \leq m \leq k$ **do**
          $U \leftarrow \mathbb{I}\left((\tilde{\mathcal{B}} \cap \mathcal{B}_{k,i}); \Gamma_m\right)$
          $\bar{\Gamma}_m = \Gamma_{m,U}$
        end for
        $Q \leftarrow \mathbb{I}\left(\tilde{\mathcal{B}} \cap \mathcal{B}_{k,i}; \bar{\Gamma}_1, \ldots, \bar{\Gamma}_{k-1}\right) \times \{i\}$
      end while
    end if
    $k \leftarrow k - 1$
  end while
  return Q
end procedure

For any selected $(i_k, \ldots, i_M)$ if $\bigcap_{m=k}^{M} \mathcal{B}_{m,i_m} = \emptyset$, there is no need to continue with this operation. Here the proposition of Lemma 4.2.2 is being used where $\tilde{\mathcal{B}} = \mathcal{B}_{M,i_M}$. However in Algorithm 8, Equation 4.8 is not used directly. Assume that we want to find $\mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_1, \ldots, \Gamma_{M-1})$. We can write the following equation;

$$\exists (i_1, i_2, \ldots, i_{M-1}) \in \mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_1, \ldots, \Gamma_{M-1}) \iff i_m \in \mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_j) \ \forall 1 \leq m \leq M-1$$

Therefore, if we want to perform the cross-sum operation in the belief subset $\mathcal{B}_{M,i_M}$, we can first perform $\mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_m)$ for each set m independently. Assume that the pruned set at the end of this operation is shown by $\tilde{\Gamma}_j$. Then,

$$\mathbb{I}(\mathcal{B}_{M,i_M}; \tilde{\Gamma}_1, \ldots, \tilde{\Gamma}_{M-1}) = \mathbb{I}(\mathcal{B}_{M,i_M}; \Gamma_1, \ldots, \Gamma_{M-1})$$

The advantage of this step is apparent when we think of the future steps of the algorithm. For instance, at the final step of the algorithm we need to calculate $\mathbb{I}(\bigcap_{m=2}^{M} \mathcal{B}_{m,i_m}; \tilde{\Gamma}_1) = \mathbb{I}(\bigcap_{m=2}^{M} \mathcal{B}_{m,i_m}; \Gamma_1)$. As $|\tilde{\Gamma}_1| < |\Gamma_1|$, it will be easier to calculate the former case. The number of constraints inside boths LPs is the same and would be much smaller than $\sum_{m=k}^{M} N_m$, which is the result for restricted region case shown in Algorithm 7.

4.2.5 Witness Algorithm

The witness algorithm is proposed by Cassandra et al. [63]. Recall that most of the algorithms choose the vectors from the dirty set with no particular order. The main motivation of the witness algorithm is to choose the vectors that are 'close' to the vectors in the clean set. After selecting any vector $\boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$, the following set is formed.

Definition 16. *The neighboring vectors of a vector* $\boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$ *are defined as follows;*

$$\mathcal{N}(\boldsymbol{\gamma}_i) = \left\{ \sum_{m \neq k} \boldsymbol{\gamma}_{m,i_m} + \boldsymbol{\gamma}_{k,t}, \ \forall t \in Q_k \setminus \{i_k\}, \ 1 \leq k \leq M \right\} \tag{4.10}$$

The indices of vectors in set $\mathcal{N}(\boldsymbol{\gamma}_i)$ *will be denoted by* $\mathcal{IN}(\boldsymbol{\gamma}_i)$

Definition 16 gives the neighboring vectors of a vector $\boldsymbol{\gamma}_i$. A neighbor is any vector that differs by a single term in the summation over the observation set.

Algorithm 9 Witness Algorithm

```
1: procedure WITNESS(Γ_1, ..., Γ_M)
2:     Γ̄ ← Γ̄_{1:M}
3:     b ← any belief state in B
4:     γ_i ← BEST(b, Γ̄)
5:     (i_1, i_2, ..., i_M) ← index of γ in Γ̄
6:     Q ← {(i_1, i_2, ..., i_M)}
7:     F ← IN(γ)
8:     while F ≠ ∅ do
9:         (i_1, i_2, ..., i_M) ← any element in F
10:        γ ← (i_1, i_2, ..., i_M)th vector in Γ̄
11:        F ← F \ {(i_1, i_2, ..., i_M)}
12:        (δ, b) ← FNDBLF(γ, Γ̄_Q)
13:        if δ < 0 then
14:            F ← F ∪ {(i_1, i_2, ..., i_M)}
15:            γ ← BEST(b, Γ̄)
16:            (i_1, i_2, ..., i_M) ← index of γ in Γ̄
17:            F ← F ∪ IN(γ)
18:            Q ← Q ∪ {(i_1, i_2, ..., i_M)}
19:        end if
20:    end while
21:    return Γ̄_Q
22: end procedure
```

The vectors in $\mathcal{N}(\boldsymbol{\gamma}_i)$ does not necessarily share a witness point with vector $\boldsymbol{\gamma}_i$, but the set $\mathcal{N}(\boldsymbol{\gamma}_i)$ is useful due to the following theorem given in Cassandra et al. [63].

Theorem 4.2.3. *Take any vector $\boldsymbol{\gamma}_i \in \Gamma$. Define the set $\mathcal{N}(\boldsymbol{\gamma}_i)$. Then the following equation holds for all $b \in \mathcal{B}$;*

$$\boldsymbol{\gamma}_i \neq w(b, \Gamma) \iff \boldsymbol{\gamma}_i \neq w(b, \mathcal{N}(\boldsymbol{\gamma}_i) \cup \boldsymbol{\gamma}_i) \tag{4.11}$$

where the definition of $w(b, \Gamma)$ is given in Definition 6.

Proof. ($\Rightarrow$:) We know that $\exists \boldsymbol{\gamma}_k \in \Gamma$ such that $b\boldsymbol{\gamma}_k > b\boldsymbol{\gamma}_i$. We can write $\boldsymbol{\gamma}_k = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,k_m}$. As $b\boldsymbol{\gamma}_k > b\boldsymbol{\gamma}_i$, $\exists j$ such that $b\boldsymbol{\gamma}_{j,k_j} > b\boldsymbol{\gamma}_{j,i_j}$. Form $\tilde{\boldsymbol{\gamma}} = \sum_{m \neq j} \boldsymbol{\gamma}_{m,i_m} + \boldsymbol{\gamma}_{j,k_j}$. Then, $b\tilde{\boldsymbol{\gamma}} > b\boldsymbol{\gamma}_i$ and $\tilde{\boldsymbol{\gamma}} \in \mathcal{N}(\boldsymbol{\gamma}_i)$.

($\Leftarrow$:) Note that $\exists \tilde{\boldsymbol{\gamma}} \in \mathcal{N}(\boldsymbol{\gamma}_i)$ such that $b\tilde{\boldsymbol{\gamma}} > b\boldsymbol{\gamma}_i$. We know that $\tilde{\boldsymbol{\gamma}} \in \bar{\Gamma}_{1:M}$. Then $b \cdot w(b, \bar{\Gamma}_{1:M}) \geq b\tilde{\boldsymbol{\gamma}} > b\boldsymbol{\gamma}_i$. Therefore $b \notin R(\boldsymbol{\gamma}_i, \bar{\Gamma}_{1:M}) = R(\boldsymbol{\gamma}_i, \Gamma)$ ■

Theorem 4.2.3 says that if there is a belief state b where $b\boldsymbol{\gamma}_j > b\boldsymbol{\gamma}_i$ and $\boldsymbol{\gamma}_i, \boldsymbol{\gamma}_j \in \Gamma$,

then a vector from $\mathcal{N}(\boldsymbol{\gamma}_i)$ is better than vector $\boldsymbol{\gamma}_i$ at this belief point b. Neighbors of a vector can show if a vector is dominated, therefore checking for domination of vectors from the same neighborhood is a faster way to delete the dirty vectors from this neighborhood. Algorithm 9 shows how this idea is used for the selection of the dirty vectors. The main routine of the algorithm is quite similar to the Lark's routine as the selected vector is only compared to the vectors in the clean set. But for the selection of dirty vectors, priority is given to the neighbors of the clean vectors. Yet there is no guarantee for the solution of $\text{FNDBLF}(\boldsymbol{\gamma}, \bar{\Gamma}_Q)$. If the solution is at another part of the belief set, these vectors are also added to the dirty set, causing a fast increase. If a great percentage of the vectors are added to the set F at the initial steps of the algorithm, the information about the region intersection due to the cross-sum operation would not be used efficiently. To see that this algorithm is complete, the reader can refer to [44].

4.2.6 Some Other Exact Value Iteration Algorithms

There are a number of algorithms which is used for the exact value iteration. Sondik's One Pass Algorithm starts with a random belief point and the corresponding dominant vector. It then finds the set of vectors that would constrain the support set of this dominant vector, yet it was shown that the algorithm can be conservative when finding the support set of a particular vector [64]. Cheng's Linear Support algorithm [41] starts from an arbitrary belief point and finds the dominant vector at this point. Unlike Sondik's One Pass algorithm, it does not add the constraints from start but at each step narrows down the support set of the dominant vector until the actual support set of the vector is found. These algorithms do not directly aim the cross-sum operation, therefore they will not be explained in this thesis, but is mentioned here for the sake of completeness.

4.3 Using the Vector Pruning Framework for the Cross-Sum Operation

It is possible to modify the mathematical preliminaries of the vector pruning problem discussed in Section 3.3 for the cross-sum operation. As all vectors are written as constraints in Equation 4.5 of the region intersection algorithm, we can relate it to Equation 3.3 of the skyline algorithm. Similar to the LP given in Equation 3.3, every hyperplane equation is written on its own. For this, variable $y_m \geq b\boldsymbol{\gamma}_{m,i_m},\ \forall i_m \in Q_m,\ 1 \leq m \leq M$ is defined.

$$
\begin{aligned}
&\min \delta \\
&b\boldsymbol{\gamma}_{m,t_m} - y_m + x_{m,t_m} = 0, \quad \forall t_m \in Q_m,\ 1 \leq m \leq M \\
&\sum_{l=1}^{D} b(l) = 1 \\
&b(l) \geq 0, \quad \forall l \in \{1, \ldots, D\} \\
&x_{m,t_m} \geq 0, \quad \forall t_m \in Q_m,\ 1 \leq m \leq M \\
&y_m \geq 0, \quad 1 \leq m \leq M
\end{aligned}
\tag{4.12}
$$

In Equation 4.12, δ is a function that can be defined differently in different settings. Finding a belief state for $\boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$, means selecting $\delta = \sum_{m=1}^{M} x_{m,i_m}$. This is a similar LP to Equation 4.5. In Equation 4.5, the optimization for the vector $\boldsymbol{\gamma}_i = \sum_{m=1}^{M} \boldsymbol{\gamma}_{m,i_m}$ is hidden in the constraints. The constraints of Equation 4.5 are written in such a way that for any feasible solution, $x_{m,i_m} = 0,\ 1 \leq m \leq M$. As finding any of the feasible solutions is enough, $\delta = 0$.

Before discussing further the selection criterion of δ, first we want to write Equation 4.12 in matrix notation. We will first assume $M = 2$, but later show that a similar approach is valid for $M > 2$. Defining, $\mathbf{H_m} = \begin{bmatrix} \boldsymbol{\gamma}_{m,1} & \boldsymbol{\gamma}_{m,2} & \ldots & \boldsymbol{\gamma}_{m,N_m} \end{bmatrix}^T$, $\mathbf{x_m} = \begin{bmatrix} x_{m,1} & x_{m,2} & \ldots & x_{m,N_m} \end{bmatrix}^T$, $\mathbf{x} = \begin{bmatrix} \mathbf{x_1}^T \mathbf{x_2}^T \end{bmatrix}^T$, $\mathbf{y} = \begin{bmatrix} y_1 & y_2 \end{bmatrix}^T$ Equation 4.12

is written in matrix form;

$$\min z = \mathbf{cx}$$

$$\begin{bmatrix} \mathbf{H}_1 & -\mathbf{e} & 0 & \mathbf{I} & 0 \\ \mathbf{H}_2 & 0 & -\mathbf{e} & 0 & \mathbf{I} \\ \mathbf{e}^T & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b^T \\ y_1 \\ y_2 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \tag{4.13}$$

where $\mathbf{0}$ is the zero vector, $\mathbf{e} = [1 \dots 1]^T$, and $\mathbf{I}$ is the identity matrix of relevant sizes. The selection of $\mathbf{c}$ depends on our purpose which is represented by δ in Equation 4.12. Note that, there are a total of $D + \sum_{m=1}^{2} N_m + 2$ variables in the simplex tableau; D coordinate variables b, 2 variables y_m and $\sum_{m=1}^{2} N_m$ slack variables $x_{m,l}$.

Definition 17. *At any point of the simplex iterations, the active vectors from the set Γ_m are shown with $\Gamma_{m,A}$ where $D_m := |\Gamma_{m,A}|$. Moreover, $\Gamma_A = \bigcup_m \Gamma_{m,A}$.*

Without loss of generality, at any point of the simplex iterations, assume that $\Gamma_{m,A} = \{\boldsymbol{\gamma}_{m,j}\}_{j=1}^{D_m}$. Similar to Chapter 3, we will assume vectors inside $\Gamma_{m,A}$ are linearly independent to avoid degenerate solutions. Furthermore, any of the D vectors from the set Γ_A are also assumed to be independent. We will partition each matrix $\mathbf{H_m}$ into $\mathbf{H_{m,A}} = \begin{bmatrix} \boldsymbol{\gamma}_{m,1} & \boldsymbol{\gamma}_{m,2} & \dots & \boldsymbol{\gamma}_{m,D_m} \end{bmatrix}^T$ and $\mathbf{H_{m,N}} = \begin{bmatrix} \boldsymbol{\gamma}_{m,D_m+1} & \boldsymbol{\gamma}_{m,D_m+2} & \dots & \boldsymbol{\gamma}_{m,N_m} \end{bmatrix}^T$. We obtain the following simplex tableau that includes all of the constraints as well as the right hand side of the equations as the last column. The final row represents the objective function of the optimization where $\mathbf{c} = [\mathbf{c}_{1,\mathbf{A}}\ \mathbf{c}_{1,\mathbf{N}}\ \mathbf{c}_{2,\mathbf{A}}\ \mathbf{c}_{2,\mathbf{N}}]$.

$$
\left[\begin{array}{c|ccccccc}
b & y_1 & y_2 & \mathbf{x}_{1,\mathbf{A}} & \mathbf{x}_{1,\mathbf{N}} & \mathbf{x}_{2,\mathbf{A}} & \mathbf{x}_{2,\mathbf{N}} & z \\
\hline
\mathbf{H}_{1,\mathbf{A}} & -\mathbf{e} & 0 & \mathbf{I} & 0 & 0 & 0 & 0 \\
\mathbf{H}_{1,\mathbf{N}} & -\mathbf{e} & 0 & 0 & \mathbf{I} & 0 & 0 & 0 \\
\mathbf{H}_{2,\mathbf{A}} & 0 & -\mathbf{e} & 0 & 0 & \mathbf{I} & 0 & 0 \\
\mathbf{H}_{2,\mathbf{N}} & 0 & -\mathbf{e} & 0 & 0 & 0 & \mathbf{I} & 0 \\
\mathbf{e}^T & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\hline
0 & 0 & 0 & \mathbf{c}_{1,\mathbf{A}} & \mathbf{c}_{1,\mathbf{N}} & \mathbf{c}_{2,\mathbf{A}} & \mathbf{c}_{2,\mathbf{N}} & 0
\end{array}\right]
$$

$$
\equiv \qquad (4.14)
$$

$$
\left[\begin{array}{c|ccccccc}
b & y_1 & y_2 & \mathbf{x}_{1,\mathbf{N}} & \mathbf{x}_{2,\mathbf{N}} & \mathbf{x}_{1,\mathbf{A}} & \mathbf{x}_{2,\mathbf{A}} & z \\
\hline
\mathbf{H}_{1,\mathbf{A}} & -\mathbf{e} & 0 & 0 & 0 & \mathbf{I} & 0 & 0 \\
\mathbf{H}_{2,\mathbf{A}} & 0 & -\mathbf{e} & 0 & 0 & 0 & \mathbf{I} & 0 \\
\mathbf{e}^T & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\mathbf{H}_{1,\mathbf{N}} & -\mathbf{e} & 0 & \mathbf{I} & 0 & 0 & 0 & 0 \\
\mathbf{H}_{2,\mathbf{N}} & 0 & -\mathbf{e} & 0 & \mathbf{I} & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & \mathbf{c}_{1,\mathbf{N}} & \mathbf{c}_{2,\mathbf{N}} & \mathbf{c}_{1,\mathbf{A}} & \mathbf{c}_{2,\mathbf{A}} & 0
\end{array}\right]
$$

The second matrix in Equation 4.14 is obtained by elementary row and column operations. The sizes of the identity matrices are not same as well as the sizes of the zero matrices, but each can be found from the structure of the matrix. Discarding the objective function, LP multiplies the second matrix of Equation 4.14 by the inverse of the following part of it.

$$
\mathbf{S} = \begin{bmatrix} \mathbf{S}_\mathbf{A} & 0 \\ \mathbf{S}_\mathbf{N} & \mathbf{I} \end{bmatrix} \qquad (4.15)
$$

where $\mathbf{S}_\mathbf{A} = \begin{bmatrix} \mathbf{H}_{1,\mathbf{A}} & -\mathbf{e} & 0 \\ \mathbf{H}_{2,\mathbf{A}} & 0 & -\mathbf{e} \\ \mathbf{e}^T & 0 & 0 \end{bmatrix}$ and $\mathbf{S}_\mathbf{N} = \begin{bmatrix} \mathbf{H}_{1,\mathbf{N}} & -\mathbf{e} & 0 \\ \mathbf{H}_{2,\mathbf{N}} & 0 & -\mathbf{e} \end{bmatrix}$.

Note that $\mathbf{S}$ is invertible if $\mathbf{S}_\mathbf{A}$ is invertible. The column rank of matrix $\mathbf{S}_\mathbf{A}$ is $D+2$ which should be equal to its row rank. Therefore, it is possible to satisfy at most $D+1$ of the $y_m \geq b\boldsymbol{\gamma}_{m,j},\ \forall j \in Q_m,\ 1 \leq m \leq 2$ type of constraints to be active (discarding degeneracies). In other words, $|\Gamma_A| = \sum_{m=1}^{2} D_m = D+1$.

Lemma 4.3.1. *Take* $\Gamma_{i,A},\ 1 \leq i \leq 2$. *Assume that the set of equations*

$$\mathbf{S_A} \begin{bmatrix} b^T \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ 1 \end{bmatrix}$$

has a solution $[b_A\ y_{1,A}\ y_{2,A}]^T$. *Then,* $\mathbf{S_A}$ *is invertible.*

Proof. To use the results of Lemma 3.3.1, define $\mathbf{H_A} = \begin{bmatrix} \mathbf{H_{1,A}} & -\mathbf{e} \\ \mathbf{H_{2,A}} & 0 \end{bmatrix}$. Note that $\mathbf{H_A}$ is a square matrix and the selection of the vectors in $\mathbf{H_{m,A}}$, $m \in \{1, 2\}$ is determined by the linear programming iterations. Therefore, we can safely assume that this matrix is invertible. Then taking $\mathbf{h} = \mathbf{H_A}^{-1}\mathbf{e}$ and multiplying both sides of the equation in the lemma by the row vector $[\mathbf{h}\ -1]$ from left we can see that the last row of $\mathbf{S_A}$ is linearly independent to the other rows and therefore, $\mathbf{S_A}$ is invertible. ■

Using Lemma 4.3.1, we can write the simplex tableau as follows;

$$\left[\begin{array}{ccc} \mathbf{S_A} & 0 & \mathbf{I} \\ \mathbf{S_N} & \mathbf{I} & 0 \\ \hline 0 & \mathbf{c_N} & \mathbf{c_{A,0}} \end{array}\right] \sim \left[\begin{array}{ccc} \mathbf{I} & 0 & \mathbf{S_A}^{-1} \\ 0 & \mathbf{I} & -\mathbf{S_N}\mathbf{S_A}^{-1} \\ \hline 0 & \mathbf{c_N} & \mathbf{c_{A,0}} \end{array}\right] = \mathbf{P} \tag{4.16}$$

where $\mathbf{c_N} = [\mathbf{c_{1,N}}\ \mathbf{c_{2,N}}]$, $\mathbf{c_A} = [\mathbf{c_{1,A}}\ \mathbf{c_{2,A}}]$ and $\mathbf{c_{A,0}} = [\mathbf{c_A}\ 0]$. The final zero of the row vector $\mathbf{c_{A,0}}$ stands for the right hand side of the simplex tableau. We finally define $\mathbf{c} = [\mathbf{c_N}\ \mathbf{c_A}]$ which shows the coefficient vector of the objective function.

Up to this point, we have not discussed the selection of the objective function $z = \mathbf{cx}$ in Equation 4.13. Before the formal presentation the selection procedure is explained with an example. Assume that $D = 2$ and we have two set of vectors $\Gamma_1 = \{[4\ 0]^T, [0\ 4]^T\}$ and $\Gamma_2 = \{[1\ 0]^T, [0.1\ 0.3]^T\}$. We want to calculate $\Gamma_{1:2}$. Table 4.1 shows the simplex tableau written by using Equation 4.13. Our aim is to find if $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,1}$ is a non-dominated vector. Therefore the objective function is written as min $x_{1,1} + x_{2,1}$. Table 4.1 shows the simplex iteration where $\Gamma_A = \{\boldsymbol{\gamma}_{1,1}, \boldsymbol{\gamma}_{1,2}, \boldsymbol{\gamma}_{2,1}\}$. The solution occurs at the intersection point of the hyperplanes formed by vectors $\boldsymbol{\gamma}_{1,1}$ and $\boldsymbol{\gamma}_{1,2}$. The hyperplane equation formed by $\boldsymbol{\gamma}_{2,1}$ is used

Table 4.1: The solution for $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,1} \in \Gamma_1 \oplus \Gamma_2$ by region intersection, $D = 2$

	$b(1)$	$b(2)$	y_1	y_2	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	$x_{2,2}$	RHS
$\boldsymbol{\gamma}_{1,1}$	4	0	-1	0	1	0	0	0	0
$\boldsymbol{\gamma}_{1,2}$	0	4	-1	0	0	1	0	0	0
$\boldsymbol{\gamma}_{2,1}$	1	0	0	-1	0	0	1	0	0
$\boldsymbol{\gamma}_{2,2}$	0.1	0.3	0	-1	0	0	0	1	0
SC	1	1	0	0	0	0	0	0	1
z	0	0	0	0	1	0	1	0	0

$\Downarrow$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	4	0	-1	0	0	1	0	0	0
$\boldsymbol{\gamma}_{1,2}$	0	4	-1	0	0	0	1	0	0
$\boldsymbol{\gamma}_{2,1}$	1	0	0	-1	0	0	0	1	0
SC	1	1	0	0	0	0	0	0	1
$\boldsymbol{\gamma}_{2,2}$	0.1	0.3	0	-1	1	0	0	0	0
z	0	0	0	0	0	1	0	1	0

$\Downarrow$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	1	0	0	0	0	1/8	-1/8	0	0.5
$\boldsymbol{\gamma}_{1,2}$	0	1	0	0	0	-1/8	1/8	0	0.5
$\boldsymbol{\gamma}_{2,1}$	0	0	1	0	0	-0.5	-0.5	0	2
SC	0	0	0	1	0	1/8	-1/8	-1	0.5
$\boldsymbol{\gamma}_{2,2}$	0	0	0	0	1	0.15	-0.15	-1	0.3
z	0	0	0	0	0	1	0	1	0

for determining the value of y_2. As the selected $\mathbf{S_A}$ matrix is invertible and both $\boldsymbol{\gamma}_{1,1}$ and $\boldsymbol{\gamma}_{2,1}$ are active vectors in this solution, the requirement to minimize $x_{1,1} + x_{2,1}$ is satisfied at point $b = [0.5\ 0.5]$ as $x_{1,1} + x_{2,1} = 0$. Note that, this point is a witness point for all vectors $\boldsymbol{\gamma} \in \bigoplus_m \Gamma_{m,A}$. That means vector $\boldsymbol{\gamma}_{1,2} + \boldsymbol{\gamma}_{2,1}$ is also active as $x_{1,2} = 0$ for our case.

After reaching the solution $b = [0.5\ 0.5]$ for $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,1}$ at Table 4.1, we check the set of dirty vectors and see that the dirty set contains the vectors $\boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,2},\ i_1 \in \{1, 2\}$. Note that these two vectors are neighbors of the active vectors at $b = [0.5\ 0.5]$. For both of these cases, we need to set $c_{2,2} = 1$. For the moment, assume that we have not decided on the values of $c_{1,1}$ and $c_{1,2}$ as shown in Table 4.2. We know that there is a region where $x_{2,2} = 0$ because Γ_2 is a minimal set. Therefore, when the objective function is set to min $x_{2,2}$, that is, when $c_{1,1}$ and $c_{1,2}$ are both zero, there must be at least one negative coefficient in the objective function. It can be seen from the first part of Table 4.2 that the coefficient of

$x_{1,1}$ is less than zero. From the second simplex tableau in Table 4.2, we can see that $x_{1,i_1},\ i_1 \in \{1,2\}$ are the slack variables of the active constraints. Therefore, having them in the objective function only affects the values of $c_{1,i_1},\ i_1 \in \{1,2\}$.

Table 4.2: New search at $b = [0.5\ 0.5]$ for $\boldsymbol{\gamma}_{1,i_1} + \boldsymbol{\gamma}_{2,2}$ by region intersection, $D = 2$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	1	0	0	0	0	1/8	-1/8	0	0.5
$\boldsymbol{\gamma}_{1,2}$	0	1	0	0	0	-1/8	1/8	0	0.5
$\boldsymbol{\gamma}_{2,1}$	0	0	1	0	0	-0.5	-0.5	0	2
SC	0	0	0	1	0	1/8	-1/8	-1	0.5
$\boldsymbol{\gamma}_{2,2}$	0	0	0	0	1	0.15	-0.15	-1	0.3
z	0	0	0	0	1	$c_{1,1}$	$c_{1,2}$	0	0

$\Downarrow$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	1	0	0	0	0	1/8	-1/8	0	0.5
$\boldsymbol{\gamma}_{1,2}$	0	1	0	0	0	-1/8	1/8	0	0.5
$\boldsymbol{\gamma}_{2,1}$	0	0	1	0	0	-0.5	-0.5	0	2
SC	0	0	0	1	0	1/8	-1/8	-1	0.5
$\boldsymbol{\gamma}_{2,2}$	0	0	0	0	1	0.15	-0.15	-1	0.3
z	0	0	0	0	0	$c_{1,1} -$ 0.15	$c_{1,2} +$ 0.15	1	-0.3

Now we can decide on the values of $c_{1,1}$ and $c_{1,2}$. When $c_{1,1} = 1$, we are searching for the vector $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$. From the second part of Table 4.2, we can see that the optimization ends when $c_{1,1} = 1$. Therefore, we can eliminate the vector $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$. So by only checking the coefficients of $b\boldsymbol{\gamma}_{2,2} - y_2 + x_{2,2} = 0$, we were able to make a decision about the neighboring vectors of the active vectors that contain $\boldsymbol{\gamma}_{2,2}$.

For Table 4.2, taking $c_{1,1} = 1$ was enough to end the optimization for the vector $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$ as $c_{1,1} - 0.15 = 0.85 > 0$. However this depends on the coefficients of the non-basic variables of the constraint $b\boldsymbol{\gamma}_{2,2} - y_2 + x_{2,2} = 0$. Now, we will approach the problem in a different way. Recall that $\boldsymbol{\gamma}_{1,1}$ is an active vector in the last simplex tableau of Table 4.2 and our aim is to find a witness point for $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$. This means to find a solution where $x_{1,1} + x_{2,2} = 0$. Note that it is also possible to reach the same solution if we were to take $c_{1,1}x_{1,1} + x_{2,2} = 0$ where $c_{1,1} \gg 1$. Then take $c_{1,1} = \infty$. This is equivalent to keeping constraint $b\boldsymbol{\gamma}_{1,1} - y_1 + x_{1,1} = 0$ active while reaching the witness point of $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$. This

Table 4.3: Searching a solution for $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$ by fixing $x_{1,1} = 0$, $D = 2$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,1}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	1	0	0	0	0	1/8	-1/8	0	0.5
$\boldsymbol{\gamma}_{1,2}$	0	1	0	0	0	-1/8	1/8	0	0.5
$\boldsymbol{\gamma}_{2,1}$	0	0	1	0	0	-0.5	-0.5	0	2
SC	0	0	0	1	0	1/8	-1/8	-1	0.5
$\boldsymbol{\gamma}_{2,2}$	0	0	0	0	1	0.15	-0.15	-1	0.3
z	0	0	0	0	1	$c_{1,1}$	0	0	0

$\Downarrow$

	$b(1)$	$b(2)$	y_1	y_2	$x_{2,2}$	$x_{1,2}$	$x_{2,1}$	RHS
$\boldsymbol{\gamma}_{1,1}$	1	0	0	0	0	-1/8	0	0.5
$\boldsymbol{\gamma}_{1,2}$	0	1	0	0	0	1/8	0	0.5
$\boldsymbol{\gamma}_{2,1}$	0	0	1	0	0	-0.5	0	2
SC	0	0	0	1	0	-1/8	-1	0.5
$\boldsymbol{\gamma}_{2,2}$	0	0	0	0	1	-0.15	-1	0.3
z	0	0	0	0	0	0.15	0	-0.3

operation is shown in Table 4.3. The important observation here is that when we force $\boldsymbol{\gamma}_{1,1}$ to be active in Table 4.3, there is no vector left in set Γ_1 that we can select to move. From the second simplex tableau in Table 4.3, we can see that the optimization for the vector $\boldsymbol{\gamma}_{1,1}+\boldsymbol{\gamma}_{2,2}$ has come to an end. As $\boldsymbol{\gamma}_{2,2}$ is not an active vector at this point, we can see that vector $\boldsymbol{\gamma}_{1,1} + \boldsymbol{\gamma}_{2,2}$ is dominated. This observation leads us to the following lemma.

Lemma 4.3.2. *Assume that in Equation 4.16, $b\boldsymbol{\gamma}_{m,i_m} - y_j + x_{m,i_m} = 0$ is an active constraint. For finding the vectors in $\mathbb{I}(\mathcal{B}_m^{i_m}; \Gamma_1, \ldots, \Gamma_{m-1}, \Gamma_{m+1}, \ldots, \Gamma_M)$, we can delete the column corresponding to x_{m,i_m}.*

Proof. As $b\boldsymbol{\gamma}_{m,i_m} - y_m + x_{m,i_m} = 0$ is an active constraint $x_{m,i_m} = 0$, the row pertaining to the variable x_{m,i_m} correspond the third column block of the matrix $\mathbf{P}$ in Equation 4.16. If we omit the column corresponding to x_{m,i_m}, we force the equation $b\boldsymbol{\gamma}_{m,i_m} + y_m = 0$ to be satisfied in every simplex iteration. ■

Then, similar to Lemma 3.3.2, this idea can be used to prune vectors as follows;

Lemma 4.3.3. *For any simplex iteration given in Equation 4.16, take any vector $\boldsymbol{\gamma}_{j,k} \notin \Gamma_A$. Define the vector $\boldsymbol{\gamma} = \sum_{m \neq j} \boldsymbol{\gamma}_{m,i_m} + \boldsymbol{\gamma}_{j,k}$ where $\boldsymbol{\gamma}_{m,i_m} \in \Gamma_{m,A}, \forall m \neq j$. Then $\boldsymbol{\gamma}$ is dominated, if the constraint row $b\boldsymbol{\gamma}_{j,k} - y_j + x_{j,k} = 0$ have all negative coefficients except for the coefficients of the non-basic slacks x_{m,i_m}, $\forall m \neq j$.*

Proof. We know that $x_{j,k}$ belongs to one of the inactive vectors in Equation 4.16. Therefore for the objective function $\mathbf{cx}$ if we take $c_{j,k} = 1$ then $\mathbf{c_N} \neq 0$. Assume that the only nonzero entry of $\mathbf{c_N}$ corresponds to the slack variable of $b\boldsymbol{\gamma}_{j,k} - y_j + x_{j,k} = 0$. This constraint is represented as one of the rows in the second row block of $\mathbf{P}$ in Equation 4.16. This row is in the form $\mathbf{r}_1 = \begin{bmatrix} \mathbf{0} & \mathbf{e_{j,k}}^T & -\tilde{\boldsymbol{\tau}}_{j,k} & x_{j,k} \end{bmatrix}$ where $\mathbf{e_{j,k}}$ is a unit vector with the only nonzero entry corresponding to the coefficient of $x_{j,k}$ and the last entry of $\mathbf{r}_1$ corresponds to the value of the slack variable for this feasible solution. As $\boldsymbol{\gamma}_{j,k}$ is an inactive vector, $x_{j,k} > 0$. The row vector $-\tilde{\boldsymbol{\tau}}_{j,k}$ represents the coefficients of the non-basic variables for this particular solution.

Now take $\boldsymbol{\gamma}_j$. For finding a witness point for this vector, write the objective function min $\mathbf{cx} = \sum_{m \neq j} c_{m,i_m} x_{m,i_m} + x_{j,k}$. Here we have taken $c_{j,k} = 1$. As we know that $\boldsymbol{\gamma}_{m,i_m} \in \Gamma_A$, $\forall m \neq j$, the objective function is of the form $\mathbf{r}_2 = \begin{bmatrix} \mathbf{0} & \mathbf{e_{j,k}}^T & \mathbf{c_A} & 0 \end{bmatrix}$ where the only non-zero entries of $\mathbf{c_A}$ are c_{m,i_m} $\forall m \neq j$. $\mathbf{c_A}$ is defined in Equation 4.16.

For finding the value of the objective function in this particular solution, it is necessary to subtract $\mathbf{r}_1$ from $\mathbf{r}_2$. The resulting row is in the form $\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{c_A} + \tilde{\boldsymbol{\tau}}_{j,k} & 0 \end{bmatrix}$. It can be seen that the only nonzero entries correspond to the coefficients of the non-basic variables. From Lemma 4.3.2, to stay at $\mathcal{B}_{m,i_m}$ means taking $c_{m,i_m} = \infty$ or equivalently discarding the column corresponding to the non-basic variable x_{m,i_m} from the simplex tableau. Therefore staying at $\bigcap_{m \neq j} \mathcal{B}_{m,i_m}$ means to discard the columns corresponding to non-zero entries of $\mathbf{c_A}$. For the zero entries of $\mathbf{c_A}$ if $\tilde{\boldsymbol{\tau}}_{j,k}$ has all non-negative coefficients, we can conclude that the optimization for min $\mathbf{cx}$ ends at this point. In mathematical terms we have proven that, $\bigcap_{m \neq j} \mathcal{B}_{m,i_m} \neq \emptyset$ but $\bigcap_{m \neq j} \mathcal{B}_{m,i_m} \bigcap \mathcal{B}_{j,k} = \emptyset$. Therefore $\boldsymbol{\gamma}$ cannot be a non-dominated vector. ■

Lemma 4.3.3 shows that for any selection of Γ_A, we start with a sign check of the block matrix $\mathbf{S_N S_A}^{-1}$. Assume that a row of $\mathbf{S_N S_A}^{-1}$ have dominantly negative coefficients and this row is related to the constraint $b\boldsymbol{\gamma}_{j,k} - y_j + x_{j,k} = 0$. This means that the cross-sum of vector $\boldsymbol{\gamma}_{j,k}$ with the vectors inside the current active set Γ_A would likely to be dominated. Therefore, for all inactive vectors $\boldsymbol{\gamma}_{j,k}$ we

can see if $\boldsymbol{\gamma} = \sum_{m \neq j} \boldsymbol{\gamma}_{m,i_m} + \boldsymbol{\gamma}_{j,k}$ where $\boldsymbol{\gamma}_{m,i_m} \in \Gamma_{m,A}, \forall m \neq j$ is dominated by just checking the corresponding row of $\mathbf{S_N S_A}^{-1}$.

4.4 Cross-Sum Pruning with Multiple Objective Functions

Algorithm 10 Cross-Sum Pruning with Multiple Objective Functions

procedure CSwM($\{\Gamma_m\}_{m=1}^M$)
  $\bar{\Gamma} \leftarrow \bar{\Gamma}_{1:M}$, $F \leftarrow Q_{1:M}$, $Q \leftarrow \emptyset$
  $N = |\bar{\Gamma}|$, $\mathbf{H} = \begin{bmatrix} \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \dots & \boldsymbol{\gamma}_N \end{bmatrix}$
  $b_1(1) = 1, j \leftarrow \arg\max_{k \in F} b_1 \boldsymbol{\gamma}_k$, $i \leftarrow \arg\min_{k \in F} b_1 \boldsymbol{\gamma}_k(1)$
  $F \leftarrow F \setminus \{j\}$, $Q \leftarrow \{j\}$
  $\mathbf{P} \leftarrow$ LPINIT($\{\Gamma_m\}_{m=1}^M$)
  $t \leftarrow 0$
  while $F \neq \emptyset$ **do**
    $\mathbf{P} \leftarrow$ LPOBJSET($\mathbf{P}, i$)
    while $i \in F$ **do**
      $t \leftarrow t + 1$, $\Gamma_A \leftarrow \emptyset$
      $(b_t, y_t, \mathbf{P}) \leftarrow$ LPITER1($\mathbf{P}$)
      for all $x_{m,k} == 0$ **do**
        $\Gamma_{m,A} \leftarrow \Gamma_{m,A} \cup \{\boldsymbol{\gamma}_{m,k}\}$
      end for
      for all $\boldsymbol{\gamma}_j \in \bigoplus_m \Gamma_{m,A}$ **do**
        if $j \notin Q$ **then**
          $Q \leftarrow Q \cup \{j\}$, $F \leftarrow F \setminus \{j\}$
        end if
      end for
      for all $x_{j,k} \neq 0$ **do**
        for all $\boldsymbol{\gamma}_l \in \bigoplus_{m \neq j} \Gamma_{m,A} \oplus \{x_{j,k}\}$ **do**
          $(l_1, l_2, \dots, l_M) \leftarrow$ index of $\boldsymbol{\gamma}_l$
          $\tau \leftarrow \tilde{\boldsymbol{\tau}}_{j,k}$
          Remove the coefficients of x_{m,l_m}, $\forall m \neq j$ from τ
          if $\tau \geq 0$ **then**
            $F \leftarrow F \setminus \{l\}$
          end if
        end for
      end for
    end while
    $\mathbf{x} = b_t \mathbf{H} - y_t$
    $i \leftarrow \arg\min_{k \in F} \mathbf{x}(k)$
  end while
  return $\bar{\Gamma}_Q$
end procedure

Algorithm 10 is the adaptation of the multiple objection function idea to the

Subroutines for the Cross-Sum Pruning with Multiple Objective Functions

procedure LPINIT($\{\Gamma_m\}_{m=1}^{M}$)
  write the initial simplex tableau $\mathbf{P}$
  min 0 subject to
  $b\boldsymbol{\gamma}_{m,t_m} - y_m + x_{m,t_m} = 0, \quad \forall t_m \in Q_m,\ 1 \leq m \leq M$
  $\sum_{l=1}^{D} b(l) = 1$
  $b \geq 0, x_{m,t_m} \geq 0, y_m > 0$
  return $\mathbf{P}$
end procedure
procedure LPITER1($\mathbf{P}$)
  do one primal simplex iteration to $\mathbf{P}$
  return $(b_t, y_t, \mathbf{P})$
end procedure
procedure LPOBJSET($\mathbf{P}, k$)
  set the objective function to $\min x_k = \sum_m x_{m,k_m}$
  return $\mathbf{P}$
end procedure

cross-sum operation. The subroutines of the algorithm are also provided as a different table. Recall that in the FastCone algorithm, the aim is to find a convex cone that a dirty vector is in and narrow it down by using the other vectors in the clean set. Here the aim is to detect the vectors with non-intersecting support regions as soon as possible. The main advantage of the algorithm is the initialization and maintenance of a single simplex tableau throughout this operation. The state-of-art solution to the problem is to solve many small-sized LPs to find the vectors in some region of the belief space as discussed in restricted region algorithm or pruning by region intersection which were discussed in Algorithm 8 and 6, respectively.

In the first step of Algorithm 10 it can be seen that all possible vectors from the cross-sum operation are formed $\bar{\Gamma}_{1:M}$ and indexed by F. While the simplex tableau will only contain the vectors in the set $\{\Gamma_m\}_{m=1}^{M}$, it is not necessary to form all vectors $\bar{\Gamma}_{1:M}$. However, the indexing of possible vectors is necessary for the algorithm. i represents the vector index in set $\bar{\Gamma}_{1:M}$ and it is assumed that $x_i = \sum_m x_{m,i_m}$.

The LPINIT routine initializes the simplex tableau $\mathbf{P}$ by using all of $b\boldsymbol{\gamma}_{m,i_m} - y_m + x_{m,i_m} = 0$ type of constraints. LPITER1 is a primal simplex iteration. Contrary to FastCone, we do not need any dual simplex iterations while all

vector constraints are written to the simplex tableau at the beginning of the algorithm by LPINIT. After every LP iteration, we check the simplex tableau $\mathbf{P}$. For the non-basic variables of the simplex tableau we can find for which vectors $\boldsymbol{\gamma}_{m,k_m}$, $x_{m,k_m} = 0$ is satisfied. Clearly this means $\boldsymbol{\gamma}_{m,k_m} \in \Gamma_{m,A}$. Then any vector in the set $\bigoplus_m \Gamma_{m,A}$ is a clean vector and its index should be discarded from the dirty set F and added to Q.

Then we start checking each row of the block matrix $\mathbf{S_N S_A}^{-1}$ as discussed in Lemma 4.3.3. In the pseudocode, we have explained the procedure for every possible neighbor of the active vectors, yet it not necessary to perform check as such. Recall that for all inactive vectors $\boldsymbol{\gamma}_{j,k}$, our aim is to find the dirty vectors containing $\boldsymbol{\gamma}_{j,k}$ and eligible for pruning. For this we are allowed to delete some entries of $\tilde{\boldsymbol{\tau}}_{j,k}$, but the remaining entries should all be non-negative. Therefore it is enough to find the negative entries of $\tilde{\boldsymbol{\tau}}_{j,k}$. Each entry corresponds to a vector $\boldsymbol{\gamma}_{m,k_m}$. Any vector $\boldsymbol{\gamma}_k = \sum_m \boldsymbol{\gamma}_{m,k_m}$ which contains all of these fixed vectors can be eliminated.

The subroutine LPOBJSET sets the objective function to min $x_i = \sum_m x_{m,k_m}$. While the main elimination procedure is realized by checking the signs of the block matrix $\mathbf{S_N S_A}^{-1}$ at every step as discussed in Lemma 4.3.3, the selection of the slack of a particular vector $\boldsymbol{\gamma}_i$ guarantees that we can give a decision about this vector at the end of simplex iterations. This is necessary for the completeness of the algorithm.

4.5 Simulations

In this section, we will present the result of the experiments with artificial and benchmark problems. From the existing algorithms, we have selected two of them for comparison: Intersection Based Incremental Pruning (IBIP) and Region Based Incremental Pruning (RBIP). Since Feng et. al. [45] have shown that the selected algorithms have a superior performance to the Generalized Incremental Pruning (GIP), we have omitted GIP from our comparison. Similarly, the Witness algorithm adds the neighbors of a vector in each step to the dirty

set, which, in the worst case, would cause a computational complexity of writing every dirty vector that comes out of the cross-sum operation. Therefore for the Witness algorithm, the number of vectors inside the simplex tableau can increase exponentially compared to IBIP and RBIP, therefore it is also discarded from the comparison.

The performance of these two algorithms have been compared to Cross-Sum with Multiple Objective Functions (SCwM). The results are provided in the tables and figures below. All of the pruning algorithms and the exact value iteration algorithm are implemented in MATLAB environment. The tests are performed with a standard desktop computer (Intel Core i7-3770 3.4GHz 8GB RAM).

4.5.1 Pruning Performance of Randomly Generated Sets

To demonstrate the scalability of the proposed algorithms, we have first tested them with artificial problems, as it gives the user the flexibility to design the problem parameters as needed. We first constructed a set of random vector sets $\{\Gamma_1, \ldots, \Gamma_M\}$. Random vectors in Γ_i are created by selecting D random numbers uniformly distributed between $(0, 200)$. Then, additional random vectors are generated and added to the set provided that they are not pointwise dominated. This process is repeated until the number of vectors in Γ_i reaches n. A test problem is thus specified by the triple (M, D, n).

Notice that for any of the incremental pruning based algorithms we do not need to compute the set $\bar{\Gamma}_{1:M}$ from the beginning of the algorithm. We only would need to take the Cartesian product of the vector indices in each set Γ_i, where each resultant n-tuple represents a vector in set $\bar{\Gamma}$. Here, $M = |\Theta|$ is a substitute for the cardinality of the observation set. Increasing M means an exponential increase in the number of vectors, n^M.

Figure 4.2 shows the experimental results for 30 trials for $D = 5, 10, 15$. For comparison with 3.4 of the FastCone algorithm, we have kept the experimental parameters same. At each graph, the horizontal axis shows the initial number of dirty vectors, $|F_0| = N = 125, 625, 3125$ and the vertical axis demonstrates

the time spent by the pruning algorithms. Both axes are given in logarithmic scale. The graphs show that CSwM algorithm has a time advantage in the order of magnitude compared both to IBIP and RBIP. This is due to writing only one LP and checking for multiple objective functions at every simplex iteration.

As expected, RBIP has a better performance compared to IBIP. As the Lark's algorithm, RBIP only writes a number of vectors to the LP, therefore this results in a better performance. It can also be observed from the graphs that the performance of IBIP and RBIP become similar as the D increases. This is due to the fact that the partitioning of the belief state becomes finer and both algorithms need to calculate a number of simplex iterations for each vector considered. The advantage of using CSwM can be seen in this situation clearly while only one simplex tableau is used for all iterations.

4.5.2 Pruning Performance of Benchmark Problems

We then tested different pruning algorithms on a set of benchmark problems from [60]. The proposed algorithms are only used for comparing the performance of different cross-sum pruning algorithms for the given problem. The results can be seen in Table 4.4. The table provides the specified time horizon for each problem, h, and the time spent in cross-sum operation in the last step in milliseconds. At $h = 0$, it is assumed that there are no accumulated rewards for any of the states. Different time horizons are chosen regarding the complexity of the specific benchmark problem. As we are asked for the value function for a finite horizon, the discount factor is taken as 1 in any of the cases.

Table 4.4: Tests with benchmark problems in milliseconds

	Problem	h	time	Problem	h	time
IBIP	4x3.95	7	6.45	hanks.95	9	8.01
RBIP			4.09			1.48
CSwM			**0.218**			**0.87**
IBIP	shuttle.95	6	11.2	4x4.95	7	0.870
RBIP			5.31			0.207
CSwM			**0.134**			**0.021**

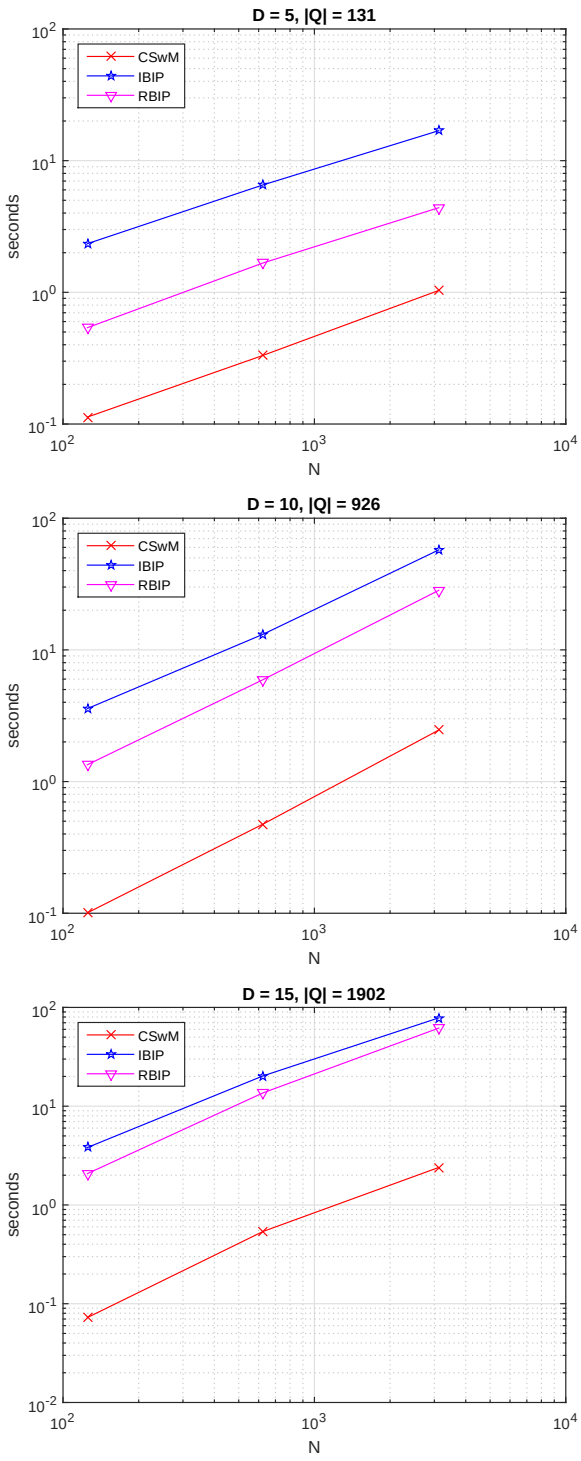


Figure 4.2: Mean time spent by different pruning algorithms (CSwM,IBIP,RBIP) for artificial problems in 30 trials. An artificial problem is defined by the triple (M, D, n). Three cases where $D = 5, 10, 15$ are examined in different graphs. Horizontal axis shows the initial number of dirty vectors $N = n^M$ and the vertical axis demonstrates the time spent by the pruning algorithms. Each graph shows three different results for $n = 5$ and $M = 3, 4, 5$. $|Q|$ shows the average number of clean vectors for $N = 5^5$ after the pruning algorithms are terminated.

For the pivoting operation, Harris ratio test is applied [47]. In the ratio test, the selection of the $\epsilon_{piv} = 10^{-8}$ value is critical especially for almost-degenerate cases where the support region of some of the vectors becomes very small. Simplex iterations also require a sign check operation to determine the pivoting column. To alleviate the roundoff error accumulation, every value smaller that $\epsilon_0 = 10^{-5}$ is assumed to be zero.

4.6 Conclusion

In any step of the exact value iteration algorithm, a set of vectors are received from the previous time step. These input vectors are first multiplied by different projection matrices, $\mathbf{P}(a)\mathbf{D}(a, o)$, resulting in multiple sets of different vectors. This operation is followed by the cross-sum addition of these sets of vectors. It is possible to represent the resultant vectors with an n-tuple, which shows the indices of the addend vectors from the vector sets entering the cross-sum operation. There is an exponential increase in the number of vectors due to the cross-sum operation and many of these vectors can be pruned.

There is a field of research that exploits the properties of the dynamic programming update steps to decrease the complexity of the LPs to be solved, many of which attack the special structure of the cross-sum operation [41–45]. The properties of the cross-sum operation has been analyzed in both the vector space [45] and belief set representation [38] of the value function. It was stated by both of these studies that for the resulting vector to be non-dominated, the vectors entering the n-tuple should have intersecting support sets. This study extends the algebraic framework for the vector pruning operation given in Chapter 3 to exploit this particular property of the cross-sum operation. The vectors with non-intersecting belief states are identified by a simple sign-check performed over the simplex tableau, and instead of writing a special LP for a selected n-tuple, the algorithm eliminates any vector when it comes across to it during this sign check.

This theoretical framework has been used for a novel algorithm called Cross-

Sum Pruning with Multiple Objective Functions. In this algorithm, the aim is to find the vectors that have non-intersecting support sets with the current active vectors in each simplex iteration. Due to the properties of the cross-sum operation, vector elimination is performed without explicitly writing all of the dirty vectors to the simplex tableau. The algorithm performance has shown to be better than the state-of-the-art algorithm by at least an order of magnitude.

CHAPTER 5

CONCLUSION

It is possible to represent the value function in partially observable Markov decision processes as a piecewise linear function if the state, action, and observation space is discrete. Exact value iteration algorithm searches for this value function by creating an exponential number of linear functions at each step, many of which can be pruned without changing the value of the value function. This pruning procedure is made possible by the use of linear programming.

This thesis is about a linear programming problem with an unusual setting. In a textbook linear programming problem, one would expect to optimize a single objective function for a given set of constraints. However, in the vector pruning problem there are as many objective functions as the number of linear constraints. There are only a few papers that specifically attack the linear programs which are being used by the exact value iteration algorithm [36,37,51]. As stated in [51], treating the LPs as a black box and focusing only on its optimal solution, results in a major performance loss. If the LP iterations are not taken into account, similar LP operations should be repeated for many of the dirty vectors.

The most important contribution of this thesis is its extensive discussion of the vector pruning problem in POMDPs from different aspects. We give an exhaustive geometric and algebraic analysis of what happens at each simplex iteration. By the geometric perspective, we mean the dual representation of the value function in belief set and vector space. The vector space representation is important, because it guides us visually in our explanation for an algorithm which focuses

on the pruning procedure. We have discussed a heuristic 'closeness' measure for the dirty vectors in the following work;

- Özgen, Selim and Mübeccel Demirekler. *A Fast Elimination Method for Pruning in POMDPs,* KI 2016, pp. 56-68.

To be able to make a formal definition of this closeness measure, this thesis uses the geometric approach to the vector pruning problem.

The algebraic framework is the explanation of this pruning operation on the corresponding simplex tableau. Our aim is to utilize the use of simplex tableau. By this, we mean an efficient procedure which would lead us to give a decision about some of the dirty vectors at every step of the simplex iterations. The second important function is maintenance. By this we mean that, at every simplex iteration the constraints inside the simplex tableau were investigated. If found unnecessary, they were let out. The theoretical endeavor has resulted in a pruning algorithm called FastCone. For a given set of clean vectors, the algorithm quickly searches for the convex region that the selected dirty vector is in and searches for another clean vector if only the current set of clean vectors is not sufficient to prune this dirty vector. Above-mentioned theoretical framework and resulting pruning algorithm has been presented in the following work;

- Özgen, Selim and Mübeccel Demirekler. *An Algebraic and Geometric Framework for Vector Pruning in POMDPs,* Journal of Artificial Intelligence Research, (submitted at May 22, 2017)

We are aware that there is an open research question for the FastCone algorithm: The number and selection of clean and dirty vectors inside the simplex tableau can be further optimized. If the number of clean vectors in the simplex tableau are small and we do not add more clean vectors to the tableau, for a selected dirty vector we will end up constraining this dirty vector to a convex cone at an infeasible vertex of the convex polyhedron, i.e., a suboptimal value of the value function. However, being at this infeasible vertex can be affordable if it can be used to delete this dirty vector. If not, it is always possible to find more

clean vectors which would guarantee that this vector stays inside a convex cone. Therefore, our priority is not about finding the clean vectors but selecting the correct combination of clean and dirty vectors inside the simplex tableau to maximize the number of pruned vectors.

Finally, we consider the cross-sum operation in this thesis as it is the most troublesome operation of the exact value iteration. This operation can be summarized as follows. For a given set of vector sets, the Cartesian product is taken, resulting in n-tuples. Each n-tuple refers to a different selection of vectors and the vectors in an n-tuple are summed up to find a new vector. There are many studies showing that for the resulting vector to be non-dominated, the vectors in the n-tuple should have intersecting support sets. This idea has been exploited by a novel algorithm called Cross-Sum Pruning with Multiple Objective Functions where the vectors with non-intersecting belief states are identified by a simple sign-check performed over the simplex tableau. The following submission is considered;

- Özgen, Selim and Mübeccel Demirekler. *Cross-Sum Pruning with Multiple Objective Functions,* IEEE Transactions on Cybernetics, (planned submission December 2017)

Both of algorithms, FastCone and Cross-Sum Pruning with Multiple Objective Functions, propose novel solutions to different stages of the exact value iteration algorithm and their efficiency has been substantiated by the experimental results.

REFERENCES

[1] M. Aoki, "Optimal control of partially observable Markovian systems," *Journal of the Franklin Institute*, vol. 280, pp. 367–386, 1965.

[2] J. J. Flokentin, "Control section: Partial observability and optimal control," *Journal of Electronics and Control*, vol. 13, pp. 263–279, sep 1962.

[3] K. Åström, "Optimal control of Markov processes with incomplete state information," *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 174–205, 1965.

[4] J. K. Satia and R. E. Lave, "Markovian decision processes with uncertain transition probabilities," *Operations Research*, vol. 21, pp. 728–740, jun 1973.

[5] R. E. Bellman, "The Theory of Dynamic Programming," *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[6] R. Bellman, "Dynamic Programming and Lagrange Multipliers," *Proceedings of the National Academy of Sciences*, vol. 42, pp. 767–769, oct 1956.

[7] R. A. Howard, *Dynamic Programming and Markov Processes.* M.I.T. Press, 1960.

[8] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Volume 1.* No. 2, Athena Scientific Belmont, MA, 2005.

[9] S. Dreyfus, "Richard Bellman on the Birth of Dynamic Programming," *Operations Research*, vol. 50, no. 1, pp. 48–51, 2002.

[10] C. Boutilier, "A POMDP Formulation of Preference Elicitation Problems," *Proceedings of AAAI Conference on AI*, pp. 239–246, 2002.

[11] J. Williams, P. Poupart, and S. . Young, "Factored partially observable Markov decision processes for dialogue management," *4th Workshop on Knowledge and Reasoning in Practical Dialogue Systems, Edinburgh.*, pp. 76–82, jan 2005.

[12] M. Mallick, V. Krishnamurthy, and B.-N. Vo, *Integrated Tracking, Classification, and Sensor Management: Theory and Applications.* John Wiley & Sons, 2012.

[13] A. O. Hero, *Foundations and Applications of Sensor Management*. Springer, 2008.

[14] J. Hoey, P. Poupart, A. von Bertoldi, T. Craig, C. Boutilier, and A. Mihailidis, "Automated handwashing assistance for persons with dementia using video and a partially observable Markov decision process," *Computer Vision and Image Understanding*, vol. 114, no. 5, pp. 503–519, 2010.

[15] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, and A. Mihailidis, "A decision-theoretic approach to task assistance for persons with dementia," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1293–1299, 2005.

[16] J. Boger, J. Hoey, P. Poupart, C. Boutilier, G. Fernie, and A. Mihailidis, "A planning system based on Markov decision processes to guide people with dementia through activities of daily living," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, pp. 323–333, apr 2006.

[17] M. Hauskrecht and H. Fraser, "Planning treatment of ischemic heart disease with partially observable Markov decision processes.," *Artificial Intelligence in Medicine*, vol. 18, no. 3, pp. 221–244, 2000.

[18] D. Hsu, W. S. Lee, and N. Rong, "A point-based POMDP planner for target tracking," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2644–2650, IEEE, may 2008.

[19] S. Temizer, M. J. M. Kochenderfer, L. L. P. Kaelbling, T. Lozano-Pérez, and J. K. Kuchar, "Collision avoidance for unmanned aircraft using Markov decision processes," in *AIAA Guidance, Navigation, and Control Conference, Toronto, Canada*, 2010.

[20] R. D. Smallwood, E. J. Sondik, and N. S. Oct, "The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon," *Operations Research*, vol. 21, pp. 1071–1088, oct 1973.

[21] G. E. Monahan, "State of the Art — A Survey of Partially Observable Markov Decision Processes : Theory , Models , and Algorithms," *Management Science*, vol. 28, pp. 1–16, jan 1982.

[22] C. H. Papadimitriou and J. N. Tsitsiklis, "The Complexity of Markov Decision Processes," *Mathematics of Operations Research*, vol. 12, pp. 441–450, aug 1987.

[23] L. K. Platzman, "Optimal Infinite-Horizon Undiscounted Control of Finite Probabilistic Systems," *SIAM Journal on Control and Optimization*, vol. 18, pp. 362–380, jul 1980.

[24] C. C. White, "A survey of solution techniques for the partially observed Markov decision process," *Annals of Operations Research*, vol. 32, no. 1, pp. 215–230, 1991.

[25] C. Boutilier and D. Poole, "Computing Optimal Policies for Partially Observable Decision Processes using Compact Representations," *Networks*, 1996.

[26] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for POMDPs," *IJCAI International Joint Conference on Artificial Intelligence*, vol. 3, pp. 1025–1030, 2003.

[27] D. Aberdeen, "A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes," *National ICT Australia, Canberra, Australia*, pp. 1–41, 2003.

[28] G. Shani, J. Pineau, and R. Kaplow, "A Survey of Point-Based POMDP Solvers," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 1–51, 2013.

[29] M. Hauskrecht, "Incremental Methods for Computing Markov Decision," in *Proceedings of AAAI Conference on AI*, 1997.

[30] T. Smith, R. Simmons, G. Shani, R. I. Brafman, and S. E. Shimony, "Heuristic Search Value Iteration for POMDPs," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 520–527, 2004.

[31] T. Smith and R. Simmons, "Point-Based POMDP Algorithms : Improved Analysis and Implementation," in *Proceedings of Annual Conference on Uncertainty in Artificial Intelligence*, pp. 542–549, jul 2005.

[32] M. T. Spaan and N. Vlassis, "Perseus: Randomized point-based value iteration for POMDPs," *Journal of Artificial Intelligence Research*, vol. 24, pp. 195–220, sep 2005.

[33] D. Braziunas, "POMDP solution methods," *University of Toronto, Tech. Rep*, 2003.

[34] M. Hauskrecht, "Value-function Approximations for Partially Observable Markov Decision Processes," *Journal of Artificial Intelligence Research*, vol. 13, no. 1, pp. 33–94, 2000.

[35] R. Munos, "Error bounds for approximate value iteration," *Proceedings of the National Conference on Artificial Intelligence*, vol. 2, pp. 1006–1011, 2005.

[36] S. Özgen and M. Demirekler, "A Fast Elimination Method for Pruning in POMDPs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9904 LNAI, pp. 56–68, Springer, 2016.

[37] E. Walraven and M. T. J. Spaan, "Accelerated Vector Pruning for Optimal POMDP Solvers," *Aaai*, pp. 3672–3678, 2017.

[38] H. Zhang, "Partially Observable Markov Decision Processes: A Geometric Technique and Analysis," *Operations Research*, vol. 58, no. 1, pp. 214–228, 2009.

[39] P. K. Ghosh, "A Unified Computational Framework for Minkowski Operations," *Computers and Graphics*, vol. 17, pp. 357–378, jul 1993.

[40] R. Schneider, *Convex Bodies: The Brunn-Minkowski Theory*. Cambridge University Press, 2013.

[41] M. L. Littman, A. R. Cassandra, and L. P. Kaelbling, "Efficient dynamic-programming updates in partially observable Markov decision processes," tech. rep., 1996.

[42] N. L. Zhang and W. Liu, "Planning in Stochastic Domains: Problem Characteristics and Approximation," Tech. Rep. Version II, Technical Report HKUST-CS96-31, Department of Computer Science, Hong Kong University of Science and Technology, 1996.

[43] A. Cassandra, M. L. Littman, and N. L. Zhang, "Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes," in *Uncertainty in Artificial Intelligence*, pp. 54–61, Morgan Kaufmann Publishers Inc., 1997.

[44] A. R. Cassandra, *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes*. PhD thesis, Brown University, 1998.

[45] Z. Feng and S. Zilberstein, "Region-Based Incremental Pruning for POMDPs," in *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pp. 146–153, AUAI Press, 2004.

[46] R. G. Bland, "New Finite Pivoting Rules for the Simplex Method," *Mathematics of Operations Research*, vol. 2, no. 2, pp. 103–107, 1977.

[47] P. M. J. Harris, "Pivot selection methods of the Devex LP code," *Mathematical programming*, vol. 5, no. 1, pp. 1–28, 1973.

[48] E. J. Sondik and N. M. Apr, "The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon : Discounted Costs," *Source: Operations Research Operations Research Society of America*, vol. 26, pp. 282–304, apr 1978.

[49] M. L. Puterman, *Markov Decision Processes.* Wiley Series in Probability and Statistics, Hoboken, NJ, USA: John Wiley & Sons, Inc., apr 1994.

[50] P. R. Kumar and P. Varaiya, *Stochastic systems: Estimation, identification and adaptive control.* Prentice-Hall, Inc., 1986.

[51] C. Raphael and G. Shani, "The Skyline algorithm for POMDP value function pruning," *Annals of mathematics and artificial intelligence*, vol. 65, no. 1, pp. 61–77, 2012.

[52] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete & Computational Geometry*, vol. 8, no. 1, pp. 295–313, 1992.

[53] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, "Computational Geometry," in *Vasa*, pp. 1–11, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[54] R. Rockafellar, *Convex Analysis.* Princeton Landmarks in Mathematics and Physics, 1996.

[55] J.-D. Boissonnat and M. Yvinec, *Algorithmic Geometry.* Cambridge University Press, 1998.

[56] T. H. Matheiss and D. S. Rubin, "A Survey and Comparison of Methods for Finding All Vertices of Convex Polyhedral Sets," 1980.

[57] M. E. Dyer, "The complexity of vertex enumeration methods," *Mathematics of Operations Research*, vol. 8, pp. 381–402, aug 1983.

[58] M. L. Littman, "The Witness Algorithm: Solving Partially Observable Markov Decision Processes," tech. rep., 1994.

[59] K. Fukuda and T. Terlaky, "Criss-cross methods: A fresh view on pivot algorithms," *Mathematical Programming*, vol. 79, pp. 369–395, oct 1997.

[60] A. Cassandra, "Tony's POMDP Page." `http://cs.brown.edu/research/ai/pomdp/index.html`, 1999.

[61] Z. Feng and S. Zilberstein, "Efficient maximization in solving POMDPs," *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, pp. 975–980, 2005.

[62] K. Fukuda, "From the zonotope construction to the Minkowski addition of convex polytopes," *Journal of Symbolic Computation*, vol. 38, pp. 1261–1272, oct 2004.

[63] A. R. Cassandra, L. P. Kaelbling, and M. L. L. B. C. Littman, "Acting optimaly in partially observable stochastic domains," tech. rep., 1994.

[64] E. C. Borera, L. D. Pyeatt, A. S. Randrianasolo, and M. Naser-Moghadasi, "POMDP filter: Pruning POMDP value functions with the Kaczmarz iterative method," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pp. 254–265, Springer, 2010.

APPENDIX A

REVISED PRUNING ALGORITHMS

In Chapter 3, we have discussed two pruning algorithms called Lark's and Iterative Skyline. For each vector in the dirty set $\bar{\Gamma}$, both algorithms construct a number of linear programs to be able to make a decision about the selected dirty vector. In this section, we propose modifications to both of these algorithms and provide the pseudocodes for the revisions done. This revision ideas are important to note, as they have led us to the FastCone algorithm given in Algorithm 4.

A.1 Iterative Skyline Algorithm with Multiple Objective Functions

Recall that for the Skyline algorithm discussed in Section 3.2.2, an constraint is defined for each vector in the set $\bar{\Gamma}$ from the beginning of the simplex iterations. As all vector constraints are already in the simplex tableau, it is possible to take any of these constraints as the objective function and check for its optimality. In this way, we are not only trying to minimize the slack variable of one vector while passing through a vertex, but we also get rid of some of the vectors from the dirty set.

Algorithm 11 is the revised version of the iterative Skyline algorithm. The main procedure is defined by ISwM, where we get an arbitrary set of vectors $\bar{\Gamma}$, and initialize an empty clean set Γ. Similar to Algorithm 3, LPINIT procedure writes the initial simplex tableau. Note that through LPINIT procedure, $\forall \boldsymbol{\gamma}_i \in \bar{\Gamma}$, a constraint $b\boldsymbol{\gamma}_i - y + x_i = 0$ is written to the LP as described by Equation 3.4. This set of equations, with the simplex constraint $\sum_l b(l) = 1$ and the non-

negativity constraints for each variable defines the simplex tableau. The same simplex tableau is used to the end of the pruning procedure.

After the simplex tableau is initialized, the objective function is selected as $\min x_i$, which is the slack variable of $\boldsymbol{\gamma}_i$ by the function LPOBJ. Yet for every simplex iteration, we call the function LPITER. LPITER is a simple simplex iteration followed by a check routine for the rows of the simplex tableau $\mathbf{P}$ at any point. This idea behind this routine was demonstrated by Lemma 3.3.2. With such a routine, it is easy to see if any of the constraints for the vectors in $\bar{\Gamma}$ are optimized. As they can never get closer to the skyline, they can be discarded from the dirty set and also from the simplex tableau $\mathbf{P}$. The algorithm continues until there are no vectors in the dirty set $\bar{\Gamma}$.

Algorithm 11 Iterative Skyline Algorithm with Multiple Objective Functions

```
1:  procedure ISwM(Γ̄)
2:      Q ← ∅, F ← {1, ..., N}
3:      P ← LPINIT(Γ̄)
4:      while F ≠ ∅ do
5:          i ← any element in F
6:          P ← LPOBJ(P, i)
7:          while i ∈ F do
8:              (P, F, Q) ← LPITER(P, F, Q)
9:          end while
10:     end while
11:     return Γ̄_Q
12: end procedure
13: procedure LPINIT(Γ)
14:     write the initial tableau P
        variables: y, x_i, b
        P(i,:) : bγ_i − y + x_i = 0, ∀γ_i ∈ Γ
        P(N+1,:) : Σ_l b(l) = 1
15:     set the objective function to zero
16:     return P
17: end procedure
18: procedure LPOBJ(P, i)
19:     set the objective function to min x_i
20:     return P
21: end procedure
22: procedure LPITER(P, F, Q)
23:     do one simplex iteration to P
24:     for all optimal P(i,:), i ∈ F do
25:         if x_i ≠ 0 then
            delete P(i,:) from the tableau
            delete i from F
26:         else
            add i to Q
            delete i from F
27:         end if
28:     end for
29:     return (P, F, Q)
30: end procedure
```

Smallwood and Sondik [20] also note the use of checking multiple objective functions for decreasing the number of linear programs. Although they note

the computational efficiency, they do not provide a discussion about how the idea is being used. The algorithm offered by Smallwood and Sondik starts by finding a clean vector $\boldsymbol{\gamma}$ and selecting a subset of the dirty vectors from the set $\bar{\Gamma}_N \subset \bar{\Gamma}$ that would allow them to traverse the corners of $R(\boldsymbol{\gamma}, \bar{\Gamma})$. Then they check at every corner point of $R(\boldsymbol{\gamma}, \bar{\Gamma})$ if the vectors in $\tilde{\boldsymbol{\gamma}} \in \bar{\Gamma}_N$ are dominated. However, the multiple objective function idea would be efficient if the vectors that would dominate $\tilde{\boldsymbol{\gamma}}$ are inside the simplex tableau and they are active. If only the dirty vectors which are used to define $R(\boldsymbol{\gamma}, \bar{\Gamma})$ are added to the simplex tableau, it is very unlikely that any of the vectors will be pruned. Moreover, a similar procedure should be repeated for all vectors in Γ for the completeness of the algorithm.

A.2 Revisions to the Lark's Algorithm

A.2.1 Sorting the vectors

For the Lark's algorithm, it does not make sense to use multiple objective functions as the algorithm starts with only one vector in the simplex tableau $\mathbf{P}$ and adds one vector to the constraint set at every LP. However, the order in which the vectors in the dirty set $\bar{\Gamma}$ are selected as the objective function can be optimized in this case. A suboptimal measure will be demonstrated in the two dimensional case.

As $b\mathbf{e} = 1$, the belief state is parametrized as $b = [\lambda \ (1-\lambda)] = f(\lambda)$. Assume that we have three non-dominated vectors as given in Figure 2.5. For these vectors, it is possible to show the following relationship:

$$\begin{aligned} \lambda = 0 &\implies \boldsymbol{\gamma}_2(2) > \boldsymbol{\gamma}_1(2) > \boldsymbol{\gamma}_0(2) \\ \lambda = 1 &\implies \boldsymbol{\gamma}_0(1) > \boldsymbol{\gamma}_1(1) > \boldsymbol{\gamma}_2(1) \end{aligned} \tag{A.1}$$

where $\boldsymbol{\gamma}_i(j)$ denotes the jth element of vector $\boldsymbol{\gamma}_i$. Therefore, $||\boldsymbol{\gamma}_0 - \boldsymbol{\gamma}_1|| < ||\boldsymbol{\gamma}_0 - \boldsymbol{\gamma}_2||$. Moreover, as $\boldsymbol{\gamma}_1$ is neighbouring to $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_2$ but $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_2$ are not neighboring, $\forall b_i \in R(\boldsymbol{\gamma}_i, \Gamma),\ i \in \{0,1,2\}$ we can assert $||b_0 - b_1|| < ||b_0 - b_2||$. Therefore non-neighboring vectors have distant support sets compared to neighboring vectors when $D = 2$.

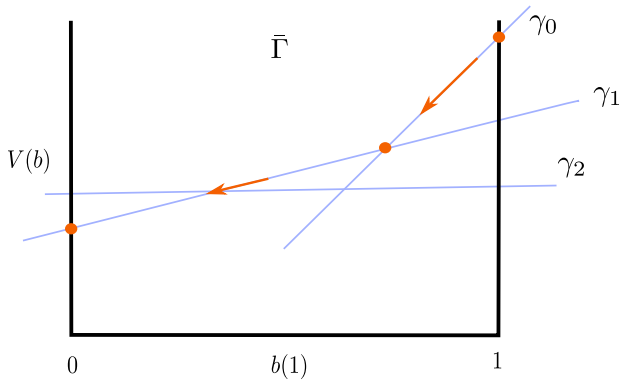


Figure A.1: The aim is to find if $\boldsymbol{\gamma}_2$ is a non-dominated vector. We start from $b \in R(\boldsymbol{\gamma}_0, \bar{\Gamma})$ and arrive at point $b(1) = 0$ in two simplex iterations. If we were to start from $b \in R(\boldsymbol{\gamma}_1, \bar{\Gamma})$, we would arrive point $b(1) = 0$ in one step. Note that $||\boldsymbol{\gamma}_0 - \boldsymbol{\gamma}_1|| < ||\boldsymbol{\gamma}_0 - \boldsymbol{\gamma}_2||$

To understand the importance of this relationship, assume that in Figure 2.5, we have only two non-dominated vectors $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_1$ in the clean set Γ, and we are trying to find if $\boldsymbol{\gamma}_2$ is a non-dominated vector. If we start from any point in the support region of $\boldsymbol{\gamma}_0$, we will first arrive at the vertex between $\boldsymbol{\gamma}_0$ and $\boldsymbol{\gamma}_1$, and then the Skyline algorithm would terminate at the maximal vertex between $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$, and Lark's algorithm would come to an end at the point $b(1) = 0$. However, if we were to start from anywhere in the support region of $\boldsymbol{\gamma}_1$, both of the pruning algorithms would arrive at the solution by visiting one vertex less.

Figure A.1 shows that if we start the pruning algorithm for a selected vector from the support set of one of its neighbouring vectors, we would reach to the solution faster. Unfortunately, it is not easy to extend this idea to the vectors of dimension greater than two. Yet the experiments show that by sorting the vectors by using the vector distance, Lark's algorithm benefit from not traversing the same vertices of the simplex tableau $\mathbf{P}$ repeatedly.

A.3 Lark's Algorithm with Initial Condition

In Definition 6, we have defined the witnessed vector for a belief state. Note that this is not an invertible function; that is if $\boldsymbol{\gamma} = w(b, \bar{\Gamma})$ we cannot say that $b \neq$

$w^{-1}(\boldsymbol{\gamma}, \bar{\Gamma})$ while the this function is one-to-many. Yet, for the Lark's algorithm (and also for any of the pruning algorithms), we find a point $b \in R(\boldsymbol{\gamma}, \bar{\Gamma})$ before deciding that vector $\boldsymbol{\gamma}$ is non-dominated. With an abuse of notation, we will call b as the wintess point of vector $\boldsymbol{\gamma}$ relative to set $\bar{\Gamma}$ and show this relation by $b = w^{\dagger}(\boldsymbol{\gamma}, \bar{\Gamma})$.

Algorithm 12 is the revised version of the Lark's algorithm. The main routine is LRwI, where we get an arbitrary set of vectors, $\bar{\Gamma}$, and initialize an empty clean set Γ. After a new vector $\boldsymbol{\gamma}$, is selected from the dirty set, the l_2 distance between this vector is compared to the vectors in the clean set Γ by the SORT routine and a vector $\hat{\boldsymbol{\gamma}}$ is selected.

As we know a witness point of vector $\hat{\boldsymbol{\gamma}}$, $b_0 = w^{\dagger}(\hat{\boldsymbol{\gamma}}, \bar{\Gamma})$, we start the LP, discussed by the LPLARK procedure, from this belief state b_0. Notice the similarity of the LPLARK procedure with the structure of the LP for the Lark's algorithm discussed in section 3.2.1. The major difference is the use of an initial condition b_0 in the algorithm.

Algorithm 12 also explains two other routines; PNTDOM and BEST. These two routines are used in the same fashion as the original algorithm. PNTDOM is used to prune, if possible, some of the dominated vectors without using linear programming and BEST is used to select one of the dominating vectors if a belief state is given. The symbol $<_{lex}$ in the pseudo-code denotes lexicographic ordering [58].

Algorithm 12 Lark's Algorithm with Initial Condition

1: **procedure** LRwI($\bar{\Gamma}$)
2: $\Gamma \leftarrow \emptyset$
3: **while** $\bar{\Gamma} \neq \emptyset$ **do**
4: $\gamma \leftarrow$ any element in $\bar{\Gamma}$
5: $b_0 \leftarrow \text{SORT}(\Gamma, \boldsymbol{\gamma})$
6: **if** PNTDOM $(\boldsymbol{\gamma}, \Gamma)$ **then**
7: $\bar{\Gamma} \leftarrow \bar{\Gamma} \setminus \{\boldsymbol{\gamma}\}$
8: **else** $(\delta, b^*) \leftarrow \text{LPLARK}(\boldsymbol{\gamma}, \Gamma, b_0)$
9: **if** $\delta < 0$ **then**
10: $\bar{\Gamma} \leftarrow \bar{\Gamma} \setminus \{\gamma\}$
11: **else**
12: $\gamma \leftarrow \text{BEST}(b^*, \bar{\Gamma})$
13: $w^\dagger(\boldsymbol{\gamma}, \bar{\Gamma}) \leftarrow b^*$
14: $\Gamma \leftarrow \Gamma \cup \{\gamma\}$
15: $\bar{\Gamma} \leftarrow \bar{\Gamma} \setminus \{\gamma\}$
16: **end if**
17: **end if**
18: **end while**
19: **return** Γ
20: **end procedure**
21: **procedure** SORT(Γ, γ)
22: $k = \infty$
23: **for all** $\hat{\boldsymbol{\gamma}} \in \Gamma$ **do**
24: **if** $k > ||\boldsymbol{\gamma} - \hat{\boldsymbol{\gamma}}||$ **then**
25: $k \leftarrow ||\boldsymbol{\gamma} - \hat{\boldsymbol{\gamma}}||$
26: $b \leftarrow w^\dagger(\hat{\boldsymbol{\gamma}}, \bar{\Gamma})$
27: **end if**
28: **end for**
29: **return** b
30: **end procedure**
31: **procedure** PNTDOM$(\boldsymbol{\gamma}, \Gamma)$
32: **for all** $\hat{\boldsymbol{\gamma}} \in \Gamma$ **do**
33: **if** $\boldsymbol{\gamma}(l) \leq \hat{\boldsymbol{\gamma}}(l), 1 \leq l \leq D$ **then**
34: **return** true
35: **end if**
36: **end for**
37: **return** false
38: **end procedure**
39: **procedure** BEST$(b, \bar{\Gamma})$
40: $\hat{\boldsymbol{\gamma}} \leftarrow \emptyset$
41: $k = -\infty$
42: **for all** $\boldsymbol{\gamma} \in \bar{\Gamma}$ **do**
43: **if** $k < b\boldsymbol{\gamma}$ **then**
44: $\hat{\gamma} \leftarrow \gamma$
45: **else**
46: **if** $k = b\boldsymbol{\gamma}$ & $\hat{\boldsymbol{\gamma}} <_{lex} \boldsymbol{\gamma}$ **then**
47: $\hat{\boldsymbol{\gamma}} \leftarrow \boldsymbol{\gamma}$
48: **end if**
49: **end if**
50: **end for**
51: **return** $\hat{\gamma}$
52: **end procedure**
53: **procedure** LPLARK$(\boldsymbol{\gamma}, \Gamma, b_0)$
54: solve the following linear program
start the linear program from b_0
variables: δ, b
$\min \delta$ subject to
$b(\boldsymbol{\gamma} - \hat{\boldsymbol{\gamma}}) + \delta > 0 \quad \forall \hat{\boldsymbol{\gamma}} \in \Gamma$
$\sum_l b(l) = 1$
$b \geq \mathbf{0}$
55: **return** (δ, b)
56: **end procedure**

APPENDIX B

CASE ANALYSIS FOR $|\Gamma_A| < D$

In Section 3.3.1, we have assumed that the active set has D vectors, however a vector can be dominated by a set of vectors such that $|\Gamma_A| < D$. Pointwise domination described in Lemma 2.4.2 is an example for this case where $|\Gamma_A| = 1$.

It can be shown that the structure of the LP is not different when $|\Gamma_A| < D$. We need to select the coordinate indexes $E \subset \{1, \ldots, D\}$ where $|A| + |E| = D$. Without loss of generality, assume that the active vectors from the set $\bar{\Gamma}$ are $\Gamma_A = \{\boldsymbol{\gamma}_i\}_{i=1}^{L}$, $L < D$ and the feasible solution is at an edge $E = \{L+1, \ldots, D\}$. Form matrices $\mathbf{H_A} = \begin{bmatrix} \boldsymbol{\gamma}_1 & \boldsymbol{\gamma}_2 & \ldots & \boldsymbol{\gamma}_L \end{bmatrix}^T$ and $\mathbf{H_N} = \begin{bmatrix} \boldsymbol{\gamma}_{L+1} & \boldsymbol{\gamma}_{L+2} & \ldots & \boldsymbol{\gamma}_N \end{bmatrix}^T$. For this selection of A and E, these matrices can be divided as $\mathbf{H_A} = \begin{bmatrix} \mathbf{H_{A,1}} & \mathbf{H_{A,2}} \end{bmatrix}$ where $\mathbf{H_{A,1}}$ is an $L \times L$ square matrix and $\mathbf{H_N} = \begin{bmatrix} \mathbf{H_{N,1}} & \mathbf{H_{N,2}} \end{bmatrix}$ where $\mathbf{H_{N,1}}$ is an $(N - L) \times L$ matrix. If E was selected differently, the order of the rows in these matrices would change accordingly. Then the simplex tableau can be arranged as follows;

$$\left[\begin{array}{cccccc} \mathbf{H_{A},_1} & \mathbf{H_{A},_2} & -\mathbf{e} & 0 & \mathbf{I} & 0 \\ \mathbf{e}^T & \mathbf{e}^T & 0 & 0 & 0 & 1 \\ \mathbf{H_{N},_1} & \mathbf{H_{N},_2} & -\mathbf{e} & \mathbf{I} & 0 & 0 \\ \hline 0 & 0 & 0 & \mathbf{c_N} & 0 & 0 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{cccccc} \mathbf{H_{A},_1} & -\mathbf{e} & 0 & \mathbf{H_{A},_2} & \mathbf{I} & 0 \\ \mathbf{e}^T & 0 & 0 & \mathbf{e}^T & 0 & 1 \\ \mathbf{H_{N},_1} & -\mathbf{e} & \mathbf{I} & \mathbf{H_{N},_2} & 0 & 0 \\ \hline 0 & 0 & \mathbf{c_N} & 0 & 0 & 0 \end{array}\right] = \mathbf{P} \tag{B.1}$$

LP multiplies $\mathbf{P}$ by the inverse of the following part of it.

$$\mathbf{S} = \begin{bmatrix} \mathbf{H_{A},_1} & -\mathbf{e} & 0 \\ \mathbf{e}^T & 0 & 0 \\ \mathbf{H_{N},_1} & -\mathbf{e} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{S_{A},_1} & 0 \\ \mathbf{S_{N},_1} & \mathbf{I} \end{bmatrix} \tag{B.2}$$

where $\mathbf{S_{A},_1} = \begin{bmatrix} \mathbf{H_{A},_1} & -\mathbf{e} \\ \mathbf{e}^T & 0 \end{bmatrix}$ and $\mathbf{S_{N},_1} = \begin{bmatrix} \mathbf{H_{N},_1} & -\mathbf{e} \end{bmatrix}$. Similar to Lemma 3.3.2, our concern is to find the dominated vectors. Therefore we will analyze the last $N - L$ rows of $\mathbf{P}$. Taking $\mathbf{h} = {\mathbf{H_{A},_1}}^{-1}\mathbf{e}$ and $\alpha = \mathbf{e}^T{\mathbf{H_{A},_1}}^{-1}\mathbf{e}$, we arrive at;

$$\mathbf{S_{N},_1}{\mathbf{S_{A},_1}}^{-1} = \begin{bmatrix} (\mathbf{H_{N},_1} + \mathbf{xh})\mathbf{e}^T{\mathbf{H_{A},_1}}^{-1} & \mathbf{x} \end{bmatrix} \tag{B.3}$$

where $\mathbf{x} = \alpha^{-1}(\mathbf{e} - \mathbf{H_{N},_1}\mathbf{h})$.

The non-zero coefficients of the third row block of $\mathbf{S}^{-1}\mathbf{P}$ are given by the following equation;

$$\begin{bmatrix} -\mathbf{S_{N},_1}{\mathbf{S_{A},_1}}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{H_{A},_2} & \mathbf{I} & 0 \\ \mathbf{e}^T & 0 & 1 \\ \mathbf{H_{N},_2} & 0 & 0 \end{bmatrix} = \begin{bmatrix} (\mathbf{H_{N},_2} + \mathbf{xe}^T - \mathbf{TH_{A},_2}) & -\mathbf{T} & \mathbf{x} \end{bmatrix} \tag{B.4}$$

where $\mathbf{T} = (\mathbf{H_N}_{,1} + \mathbf{x}\mathbf{e}^T)\mathbf{H_A}_{,1}{}^{-1}$.

Now assume that $\mathbf{H_N} = \boldsymbol{\gamma}_0^T$ and we want to find if vector $\boldsymbol{\gamma}_0$ is dominated at this corner of the unit simplex. Then,

$$\mathbf{H_N}_{,1} = \boldsymbol{\gamma}_{0,1}^T = [\boldsymbol{\gamma}_0(1) \dots \boldsymbol{\gamma}_0(L)] \tag{B.5}$$

$$\mathbf{H_N}_{,2} = \boldsymbol{\gamma}_{0,2}^T = [\boldsymbol{\gamma}_0(L+1) \dots \boldsymbol{\gamma}_0(D)] \tag{B.6}$$

$$\mathbf{x} = x_0 = \alpha^{-1}(\mathbf{e} - \boldsymbol{\gamma}_{0,1}^T \mathbf{h}) \tag{B.7}$$

$$\mathbf{T} = \tilde{\boldsymbol{\tau}}_0 = (\boldsymbol{\gamma}_{0,1}^T + x_0\mathbf{e}^T)\mathbf{H_A}_{,1}{}^{-1} \tag{B.8}$$

where x_0 is the slack variable of the hyperplane equation $b\boldsymbol{\gamma}_0 + x_0 = y$. From Lemma 3.3.3, we know that $\tilde{\boldsymbol{\tau}}_0\mathbf{e} = 1$.

Notice that $\boldsymbol{\gamma}_{0,1}$ and $\boldsymbol{\gamma}_{0,2}$ defined in Equation B.5 and B.6 is valid for the special case when $E = \{L+1, \dots, D\}$. Now, for any given E, it is possible to define the vector $\boldsymbol{\gamma}_{0,2}$ the entries of which is formed by the operation $\boldsymbol{\gamma}_{0,2}(i) = \boldsymbol{\gamma}_0(E(i))$, $1 \leq i \leq |E|$. With a slight abuse of notation we will define this newly formed vector as $\boldsymbol{\gamma}_0(E)$. Similarly, defining the set $\bar{E}$ as the complement of the set E w.r.t D, it is possible to write $\boldsymbol{\gamma}_{0,1} = \boldsymbol{\gamma}_0(\bar{E})$.

As described in Lemma 3.3.2, to give a decision about vector $\boldsymbol{\gamma}_0$, all row coefficients of the final simplex tableau should be negative. Using the right hand-side matrix at Equation B.4, we can write;

$$\tilde{\boldsymbol{\tau}}_0 > 0 \tag{B.9}$$

$$\tilde{\boldsymbol{\tau}}_0\mathbf{H_A}_{,2} - \boldsymbol{\gamma}_{0,2}^T > x_0\mathbf{e}^T \tag{B.10}$$

Lemma B.0.1. *Take Γ_A and let $E \subset \{1, \dots, D\}$ is an arbitrary set where $|A| + |E| = D$. Then it is possible to write;*

$$\tilde{\boldsymbol{\gamma}}_0 = \boldsymbol{\gamma}_0 + x_0\mathbf{e} = \sum_{j \in A} \tilde{\boldsymbol{\tau}}_0(j)\boldsymbol{\gamma}_j - \sum_{j \in E} \boldsymbol{\beta}_0(j)\mathbf{e}_j \tag{B.11}$$

Proof. Note that for given Γ_A and E, x_0 and $\tilde{\boldsymbol{\tau}}_0$ are defined uniquely by Equations B.7 and B.8, respectively. Then we can write,

$$\boldsymbol{\gamma}_0(\bar{E}) + x_0\mathbf{e} = \sum_{j \in A} \tilde{\boldsymbol{\tau}}_0(j)\boldsymbol{\gamma}_j(\bar{E}) \tag{B.12}$$

Equation B.11 is identical to Equation B.12 as $\boldsymbol{\beta}_0$ is unconstrained. ■

With Lemma B.0.1, $\boldsymbol{\gamma}_0$ is represented in terms of Γ_A for our selection of the coordinate variables E.

Definition 18. *Define the set $\Gamma_{\{A,E\}}$ where Γ_A is the set of active vectors and $E \subset \{1, \ldots, D\}$ is the index set for the coordinate variables and $|A| + |E| = D$. Then the set of equations*

$$b(E) = 0$$
$$\mathbf{S_A} \begin{bmatrix} b^T \\ y \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix}$$

has a solution $[b_A \; y_A]^T$.

Definition 19. *Define $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma^C_{\{A,E\}}$ as $\boldsymbol{\beta}_0 > 0$ and $\tilde{\boldsymbol{\tau}}_0 > 0$. This statement is equivalent to the statement of Equations B.9 and B.10.*

Theorem B.0.2. *Take Γ_A and E. The following statement is true:*

$$\left(\tilde{\boldsymbol{\gamma}}_0 \in \Gamma^C_{\{A,E\}}\right) \wedge (x_0 > 0) \implies R(\boldsymbol{\gamma}_0, \Gamma_A) = \emptyset$$

Proof. We can write for any $b = b_A + \nabla b$ such that $\nabla b \mathbf{e} = 0$. Recall that $b_A(E) = 0$, therefore $\nabla b(E) \geq 0$. Now select $k = \arg\max_i \nabla b \boldsymbol{\gamma}_i$. Then,

$$\begin{aligned}
b\boldsymbol{\gamma}_0 &= (b_A + \nabla b)\boldsymbol{\gamma}_0 \\
&= b_A\boldsymbol{\gamma}_0 + \nabla b\boldsymbol{\gamma}_0 \\
&= y_A - x_0 + \nabla b\tilde{\boldsymbol{\gamma}}_0 \\
&< y_A + \nabla b\tilde{\boldsymbol{\gamma}}_0 \\
&= y_A + \sum_{j \in A} \tilde{\boldsymbol{\tau}}_0(j)(\nabla b\boldsymbol{\gamma}_j) - \sum_{j \in E} \boldsymbol{\beta}_0(j)\nabla b e_j \\
&< y_A + \nabla b\boldsymbol{\gamma}_k \\
&= (b_A + \nabla b)\boldsymbol{\gamma}_k \\
&= b\boldsymbol{\gamma}_k
\end{aligned}$$

■

Until this point, we haven't specified the selection of the coordinate indices, E. The following theorem says that, for a given active set of vectors Γ_A, there is only one possible selection of E for $\tilde{\boldsymbol{\gamma}}_0 \in \Gamma^C_{\{A,E\}}$.

Theorem B.0.3. *Take* $\Gamma_A = \{\boldsymbol{\gamma}_i\}_{i=1}^{L}$. *Suppose that*

$$b_t(E_t) = 0$$
$$\mathbf{S_{A,1}} \begin{bmatrix} b_t(\bar{E}_t)^T \\ y_t \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \quad t \in \{1, 2\}$$

has solutions such that $b_1 \neq b_2$ *and* $|E_t| = D - L$. *Given these conditions, the following statement is true:*

$$\tilde{\boldsymbol{\gamma}}_0^1 \in \Gamma^C_{\{A,E_1\}} \implies x_0^1 < x_0^2 \wedge \tilde{\boldsymbol{\gamma}}_0^2 \notin \Gamma^C_{\{A,E_2\}}$$

Proof. As given in Lemma B.0.1, we can write for $t \in \{1, 2\}$,

$$\boldsymbol{\gamma}_0 + x_0^t \mathbf{e} = \sum_{j \in A} \tilde{\boldsymbol{\tau}}_0^t(j) \boldsymbol{\gamma}_j - \sum_{j \in E_t} \boldsymbol{\beta}_0^t(j) \mathbf{e}_j \tag{B.13}$$

We are also given that $\tilde{\boldsymbol{\tau}}_0^1 > 0$. From Definition 14, $b_2\boldsymbol{\gamma}_0 + x_0^2 = y_2$. Write,

$$b_2\boldsymbol{\gamma}_0 + x_0^1 = \sum_{j \in A} \tilde{\boldsymbol{\tau}}_0^1(j) b_2 \boldsymbol{\gamma}_j - \sum_{j \in E_1} \boldsymbol{\beta}_0^1(j) b_2 \mathbf{e}_j \tag{B.14}$$
$$= y_2 - \sum_{j \in E_1} \boldsymbol{\beta}_0^1(j) b_2 \mathbf{e}_j \tag{B.15}$$
$$< y_2 \tag{B.16}$$

Then $x_0^1 < x_0^2$. Moreover, if $\tilde{\boldsymbol{\gamma}}_0^2 \in \Gamma^C_{\{A,E_2\}}$ we do the same operations for $b_1\tilde{\boldsymbol{\gamma}}_0^2$ and arrive at $x_0^2 < x_0^1$. This is a contradiction. So, $\tilde{\boldsymbol{\gamma}}_0^2 \notin \Gamma^C_{\{A,E_2\}}$. ■

In Section 3.3.1, we have described small sized LPs, called $LP_{A,i}$, where A denotes the set Γ_A and i is the index of $\boldsymbol{\gamma}_i$. Theorem B.0.3 refers to a series of pivot operations on $LP_{A,0}$ where the initial non-basic coordinate variables are $b(E_2)$ and they are approaching to $b(E_1)$ as it is the optimal solution.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Özgen, Selim
Nationality: Turkish (TC)
Date and Place of Birth: 29/11/1983, Ankara
Marital Status: Single
Phone: 00905464745494

EDUCATION

Degree	Institution	Year of Graduation
M.A.	Boğaziçi University	2008
B.S.	Boğaziçi University	2006
High School	İzmir Science High School	2001

PROFESSIONAL EXPERIENCE

Year	Place	Enrollment
05/2017-	KIT	Research Assistant
08/2012-05/2017	METU	Research Assistant
10/2008-01/2011	Boğaziçi University	Technical Assistant

PUBLICATIONS

Journal Publications

- Özgen, Selim and Mübeccel Demirekler. *An Algebraic and Geometric Framework for Vector Pruning in POMDPs,* Journal of Artificial Intelligence Research, (submitted at May 22, 2017)

International Conference Publications

- Özgen, Selim and Mübeccel Demirekler. *A Fast Elimination Method for Pruning in POMDPs,* KI 2016, pp. 56-68.
- Özgen, Selim, Mübeccel Demirekler, and Umut Orguner. *On the solution of data association problem using rollout algorithms,* 2016 19th International Conference on Information Fusion (FUSION). IEEE, 2016.
- Özgen, Selim and Mübeccel Demirekler. *Rollout algorithms for the measurement-to-track association problem,* 2016 24th Signal Processing and Communication Application Conference (SIU), Zonguldak, 2016, pp. 1597-1600.
- Özgen, S., Sarıtaş, E., Orguner, U., Acar, D. *Delay estimation based on kinematic track information without time stamps*, 2014 22nd Signal Processing and Communication Application Conference (SIU), Trabzon, 2014, pp. 1454-1458.