Constrained Markov Decision Processes with Unknown Constraints are Feasible

Jesse Schneider
jsa378@sfu.ca

Hanyang Hu
hha160@sfu.ca

Mikhail Sinitcyn
msa242@sfu.ca

Abstract

Constrained Markov decision processes are a standard reinforcement learning framework used when it is desirable to constrain the agent's behavior, typically in order to control costs accrued by the agent. It is almost always assumed that the constraints placed upon the agent are known, yet situations do arise wherein the constraints are unknown. In this work, we consider constrained Markov decision processes with unknown constraints, and attempt to find a policy that respects the unknown constraints. Assuming only very limited feedback when constraints are violated, we show that such a policy can indeed be found via exploitation of constrained Markov decision processes' linear-programmatic structure. Furthermore, this can be done in polynomial time. We believe this result to be novel, and we discuss the methods applied to achieve this result in detail. We also present a simple example. Having done this, we consider the problem of finding a policy that not only respects the unknown constraints, but a good, or possibly even optimal, policy. With a different set of assumptions that provide more information, but still with mostly unknown constraints, we present a procedure that can find not only policies respecting the unknown constraints, but optimal policies. Furthermore, the procedure's asymptotic characteristics imply that it is more likely to find an optimal policy over time. This procedure can also be performed in polynomial time, and we believe this result to be novel as well.

1 Introduction and Motivation

Reinforcement learning is a subfield of modern artificial intelligence that has been involved in some of the most notable technical advances of the 21st century, including superhuman play in the game of Go [Silver et al., 2016], expert-level play in Atari video games [Mnih et al., 2015], autonomous driving [Kiran et al., 2021], and large language models such as GPT-4 [OpenAI, 2023, Christiano et al., 2017]. However, most of these successes have come in relatively benign situations in which the costs of erroneous actions are not significant. When there are significant potential costs, however, exploration must be managed more carefully.

Consider the following motivating example. In a telecommunications network, a variety of types of information are transmitted, including file transfers, voice calls, video, *etc.* It is typically desirable to maximize network throughput, subject to constraints on delays in the transmission of various types of data, such as live video calls [Altman, 1999]. When delays in transmission of a video call reach a certain threshold, they become intolerable. Therefore, a telecommunications network controller must experience costs when it makes decisions that have negative consequences for users. This heightened importance of costs has led to a sub-discipline of reinforcement learning that imposes costs on an agent, and attempts to make the agent operate within constraints.

Mathematically, this type of cost- and constraint-conscious reinforcement learning is explored through Markov decision processes, but with extra costs and constraints added to disincentivize costly actions. The standard framework is the constrained Markov decision process [Altman, 1999, Gu et al., 2023]. However, in the standard constrained Markov decision process formulation, the additional constraints

are assumed to be known beforehand. This is not always a realistic assumption. Indeed, there are situations involving robotic exploration [Wachi and Sui, 2020] and communications networks [Bei et al., 2015] in which constraints are *not* known in advance. For further discussion of scenarios in which constraints are unknown, primarily in cellular networks, but also in product design and experiments, see Appendix A.

In this work, we explore the relaxation of the standard assumption that the constraints in a constrained Markov decision process are known in advance, and we seek a policy that respects the unknown constraints. We show that, given limited feedback received from an oracle when a policy is proposed, such a policy can indeed be found efficiently (*i.e.*, in polynomial time). We analyze the strengths and weaknesses of our proposed method, and discuss the techniques employed to achieve this result in detail. We also present a simple example. Next, we consider the question of how to find a policy that not only respects unknown constraints, but is good, or even optimal. In Appendix F, we make a different set of assumptions that provide more information, and show that optimal policies can be found, despite mostly unknown constraints. This can also be accomplished in polynomial time.

2 Problem Formulation

A constrained Markov decision process is a tuple $(\mathcal{S}, \mathcal{A}, \rho, \mathcal{P}, r, c)$, where $\mathcal{S}$ is a finite state space, $\mathcal{A}$ is a finite action space, ρ is the starting state distribution, $\mathcal{P}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $r\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $c\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}^K$ is the cost function. The intuition for a constrained Markov decision process, as opposed to a regular Markov decision process, is that in every state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, the agent receives not only the scalar reward $r(s, a)$ but also the K scalar costs $c_1(s, a), \ldots, c_K(s, a)$. We initially assume for simplicity that all of the above are known, although we will see in Section 4 that knowledge of ρ, $\mathcal{P}$, r and c is not strictly necessary.

Given a constrained Markov decision process, the standard objective is to find a policy to maximize

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(S_t, A_t)\right],$$
$$\text{while} \quad \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t c_i(S_t, A_t)\right] \leq C_i$$

for $i = 1, \ldots, K$. The quantity to be maximized is the expected, discounted sum of rewards, and the quantities to be limited are expected, discounted sums of costs. The scalar $0 < \gamma < 1$ is the discount factor. The real numbers $C_1, \ldots, C_K$ are the constraints of the constrained Markov decision process.

In a regular constrained Markov decision process, the constraints $C_1, \ldots, C_K$ are known. We assume they are unknown, and we seek a policy that respects the unknown constraints.

In order to make progress, we make further assumptions connected with the notion of limited feedback, which is motivated further in Appendix A. In particular, we assume the existence of an oracle to which we may propose a policy. If the proposed policy is feasible, the oracle tells us Yes, and our goal is accomplished. If the proposed policy is infeasible, the oracle tells us *which* constraint was violated, but not the precise nature of the violation. If the proposed solution violates multiple constraints, we are only told which constraint was *most* violated, as measured by Euclidean distance. More details about this distance are in Appendix G.2. We assume, as is standard, that the oracle gives us the above feedback in $O(1)$ time. We call this the "furthest oracle" model.

Importantly, note that our work is reliant on the fact that a constrained Markov decision process is a linear program, and the information that we receive from the oracle concerns indices of violated constraints *of the linear program*. The constraints of the linear program are a superset of the constraints of the constrained Markov decision process. The relationship between constrained Markov decision processes and linear programs is discussed further in Section 4, and Appendix B.

For the problem formulation in which the goal is to find optimal policies, see Appendix F for a complete discussion and analysis.

3 Related Work

We are not aware of any work that considers the problem of constrained Markov decision processes with unknown constraints as we do in this work. Therefore, we will survey work that is the most related to ours. For a discussion of the applications of the work surveyed here, see Appendix C.

The textbook by Altman [1999] is the standard reference for constrained Markov decision processes, and its first three chapters provide the basic framework that we use. Unlike our work, Altman [1999] assumes the constraints are known.

The safe reinforcement learning survey by Gu et al. [2023] describes many applications of constrained Markov decision processes in safe reinforcement learning, though the constraints are assumed to be known throughout, which is unlike our work.

The work by Wachi and Sui [2020] is the only work we have encountered that considers the idea of unknown constraints in reinforcement learning. However, they use a different framework than we do, and so their work is not directly comparable to ours. Wachi and Sui [2020] consider a "safety constrained" Markov decision process with deterministic state transitions, a reward function $r\colon \mathcal{S} \to \mathbb{R}$ dependent only on the state, and a "safety function" $g\colon \mathcal{S} \to \mathbb{R}$. The general idea is that for all $s \in \mathcal{S}$, Wachi and Sui [2020] would like that $g(s) \geq h$, where h is a lower bound above which the agent is safe. Neither g nor r are completely known in advance, but it is assumed that there is an initial set of states $S_0 \subset \mathcal{S}$ such that $g(s_0) \geq h$ for all $s_0 \in \mathcal{S}_0$. From this initial set of known safe states, Gaussian processes are used to learn the safety function g and the reward function r.

Wachi and Sui [2020] do not use cumulative costs as in a regular constrained Markov decision processes, but rather a per-state safety level g. Since the framework used by Wachi and Sui [2020] is not equivalent to a standard constrained Markov decision process, the techniques that we apply in this work are not applicable to the framework used by Wachi and Sui [2020]. In particular, a regular constrained Markov decision process is equivalent to a linear program, while the framework used by Wachi and Sui [2020] is not.

Bayesian optimization is a statistically flavored type of optimization that has strong similarities to reinforcement learning [Garnett, 2023]. The language of "sequential decision-making under uncertainty" is used when optimizing a potentially unknown function f. As in [Wachi and Sui, 2020], Gaussian processes are also used. Interestingly for our purposes, the problem of unknown constraints has been considered in Bayesian optimization [Garnett, 2023]. However, Bayesian optimization does not have a concept of state, because at every time step, it is assumed that one can obtain a (potentially noisy) reading from the function f at any point in its domain $\mathcal{X}$. This is akin to assuming the ability to travel from any point in a state space $\mathcal{S}$ to any other point in a single time step, when in Markov decision processes, the agent is restricted by the transition probabilities. Therefore, the framework and results from Bayesian optimization do not appear to be immediately applicable to reinforcement learning.

The work by Zhang et al. [2022] is thematically similar to our work, although the details are different. It begins with a standard constrained Markov decision process, and then attempts to extend this framework by considering the possibility of having infinitely many constraints. The motivation for this consideration is that constraints may be of a spatial nature, *e.g.*, a constraint for every point $x \in [0, 1]^2 \subset \mathbb{R}^2$. Zhang et al. [2022] apply linear semi-infinite programming [Goberna and Lopez, 2002] to develop their theory. Nevertheless, Zhang et al. [2022] do not consider the problem of unknown constraints as we do.

Lastly, though not nominally concerned with reinforcement learning, we rely heavily on the results presented in [Bei et al., 2013].

4 Theoretical Results and Discussion

4.1 Theoretical Results

For the purposes of our work, a very important property of constrained Markov decision processes is that their solutions can be found via linear programs; for a discussion of this fact, see Appendix B. Given a constrained Markov decision process as described in Section 2, it can be solved via the following linear program:

$$
\begin{aligned}
&\text{maximize } \frac{\langle \mu, r \rangle}{1-\gamma} \\
&\text{subject to } \gamma \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mathcal{P}(s' \mid s, a) \mu(s, a) + (1-\gamma)\rho(s') = \sum_{a \in \mathcal{A}} \mu(s', a) \\
&\qquad \text{for all } s' \in \mathcal{S}, \\
&\text{and } \mu(s, a) \geq 0 \text{ for all } (s, a) \in \mathcal{S} \times \mathcal{A}, \\
&\text{and } \langle \mu, c_k \rangle \leq C_k \text{ for } k = 1, \ldots, K.
\end{aligned}
$$

The relationship between μ and the policy is discussed further in Appendix B. In order to find a feasible solution to the above linear program, despite unknown constraints $C_1, \ldots, C_K$, we rely on a convex optimization technique known as the ellipsoid method, which operates via a decreasing sequence of ellipsoids. For an introduction to the ellipsoid method, see Appendix D.

A crucial property of the ellipsoid method is that the ellipsoid method can be applied to a constrained problem *without knowledge of the problem's constraints* [Matoušek and Gärtner, 2007, Grötschel et al., 1981]. This is, in part, due to the presumed existence of an oracle that provides feedback at each step of the method.

Using the ellipsoid method, along with other techniques, methods have been developed to find a feasible solution to a linear program with unknown constraints [Bei et al., 2015]. This is the key to our ability to find a feasible solution to a constrained Markov decision process, despite not knowing its constraints.

Given a linear program with constraints of the form $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, the basic idea in the application of the ellipsoid method as in [Bei et al., 2013] is as follows. The ellipsoid method is applied to search for $(\mathbf{A}, \mathbf{b})$. At time t, the center of the current ellipsoid is $(\mathbf{A}_t, \mathbf{b}_t)$. A feasible point $\mathbf{x}$ for a linear program with constraints $\mathbf{A}_t\mathbf{x} \geq \mathbf{b}_t$ is then proposed to the oracle as a possible feasible point for the linear program with constraints defined by $(\mathbf{A}, \mathbf{b})$, which are unknown. The main source of complications occurs when the constraints defined by $(\mathbf{A}_t, \mathbf{b}_t)$ cannot be satisfied by any point. In such a case, depending on the precise nature of the unknown constraints, an alternate matrix $\mathbf{A}'_t$ can be found that is arbitrarily close to $\mathbf{A}_t$, such that the constraints defined by $(\mathbf{A}'_t, \mathbf{b}_t)$ do have a feasible point, which can be sent to the oracle. When this is not possible, more sophisticated techniques must be used. See Section 5 and Appendix G for further details.

We now state and prove our main result concerning finding feasible policies for constrained Markov decision processes with unknown constraints. This is a direct application of the methods developed in [Bei et al., 2013]. For the precise definitions of terms used in the following, and some discussion thereof, see Appendix E.

Proposition 1. *For a constrained Markov decision process with K unknown constraints, a policy satisfying the constraints can be found in time polynomial in the input size $L = O\left(|\mathcal{S}|^2 |\mathcal{A}| + |\mathcal{S}| |\mathcal{A}| K\right)$.*

Proof. Write the constrained Markov decision process as a linear program, and apply Theorem 1 from [Bei et al., 2013]. □

For completeness, Theorem 1 from [Bei et al., 2013] states that

> The UnknownLP problem can be solved in time polynomial in the input size in the furthest oracle model, provided that the input is non-degenerate.

The "UnknownLP" problem is the problem of finding a feasible point for a linear program with unknown constraints. The general method applied to find the feasible point is to apply the ellipsoid method as previously described, but there many complicating factors that must be addressed. For more discussion of the details of the methods developed in [Bei et al., 2013], see Section 5 and Appendix G.

We emphasize that the UnknownLP problem as defined in [Bei et al., 2013] is "solved" by finding a feasible point for the constraints of the linear program. It is not a solution in the regular sense of linear programming with known constraints, because it does not guarantee an *optimal* solution, which in the context of a constrained Markov decision process would be an optimal policy.

4.2 Discussion

We now present some analysis of the characteristics of the result stated in Proposition 1, and explore possible extensions. First, since the time to find a feasible policy is polynomial in the input size $L = O\left(|\mathcal{S}|^2 |\mathcal{A}| + |\mathcal{S}| |\mathcal{A}| K\right)$, we see that our method will be slower when the constrained Markov decision process has a large state space, a large action space, and/or many constraints. Furthermore, polynomial time is efficient in theory [Matoušek and Gärtner, 2007, Goldreich, 2008], not necessarily in practice (*e.g.*, matrix inversion). This leads to the question of whether it is possible to find a feasible policy in less than polynomial time. This is doubtful, since methods that are typically faster than the ellipsoid method, such as the simplex method and interior-point methods, cannot be applied to problems with unknown constraints.

In Section 2, we assumed for simplicity that we had knowledge of ρ, $\mathcal{P}$, r and c. However, as mentioned in that section, this is not strictly necessary, because the methods in [Bei et al., 2013] assume no knowledge of any of the constraints of our linear program, or indeed of the linear program's objective function.

We would like to explore the possibility of strengthening of the result presented in Proposition 1, because in reinforcement learning, typically good or optimal policies are sought, not merely feasible policies. First, we consider the possibility of finding a feasible policy more quickly if we have knowledge of ρ, $\mathcal{P}$, r and c. In this case, all that is unknown is the constraints $C_1, \ldots, C_K$, so all but K constraints of the constrained Markov decision process, considered as a linear program, are known. It would be desirable to apply this information by ensuring that each proposal $(\mathbf{A}_t, \mathbf{b}_t)$ satisfies the known portion of the constrained Markov decision process, so each proposal only risks violating the K unknown constraints. However, with this modification, we are no longer applying the ellipsoid method, since in the ellipsoid method, each new proposal is the center of a new ellipsoid that is the smallest ellipsoid containing a specified half of the old ellipsoid. Therefore, the methods developed in [Bei et al., 2013] no longer apply.

Next, we consider the possibility of strengthening the result presented in Proposition 1 by finding not just a feasible policy, but a better policy, perhaps even the optimal policy. Let us assume that we have knowledge of r, so that we know the objective function of our constrained Markov decision process, as a linear program. First, we might simply apply the method from [Bei et al., 2013] multiple times and select the result that maximizes the objective function, but this is an inefficient improvement.

Given an initial feasible point, we could compute the gradient and attempt gradient ascent to find a better solution, but given that the constraints are unknown, we have no assurance that subsequent points will satisfy the constraints when moving in the direction of the gradient. Moreover, the furthest oracle does not provide sufficient feedback to perform gradient ascent, projected into a feasible region. Lastly, we might consider starting from a feasible point and then performing a greedy search of a ball of radius ϵ around the feasible point, but this search could only be guaranteed to continue satisfying the constraints if the initial feasible point is in the interior of the constraint set to begin with, and even so, an appropriate value of ϵ cannot be known for certain.

Given these difficulties, in order to explore the possibility of finding not merely a feasible policy, but a good, or even possibly optimal policy, despite unknown constraints, we consider an alternate framework in which this is possible. A complete discussion of this alternate framework, the corresponding results and analysis, and a comparison with the present framework, are presented in Appendix F.

5 Details and Example

5.1 The Simplified Furthest Oracle Algorithm

We will provide an overview of a simplified version of the Furthest Oracle algorithm as presented in [Bei et al., 2013]. It's essential to emphasize that the version we implemented is a simplified iteration. The rationale behind this simplification will be elucidated shortly. However, it's crucial to clarify that prematurely terminating the algorithm doesn't necessarily yield optimal results. As indicated in the original paper, the full procedure must be executed to ensure a viable solution with guaranteed feasibility.

The intuition of the original Furthest Oracle algorithm is that the authors convert the algebra problem to a geometric problem. Starting with a special situation when $\mathbf{b} = 0$, the authors conceptualize the matrix $\mathbf{A}$ as a point in $\mathbb{R}^{mn}$; then based on the idea of the Ellipsoid method, they treat the known $\mathbf{A}'$ as the center of the ellipsoid, and project the rows of $\mathbf{A}'$ onto the sphere S^{n-1}, creating the resulting nearest-neighbor partition known as the Voronoi diagram[Bei et al., 2013]. By proving the fact that the index returned by the oracle precisely corresponds to the furthest Voronoi cell containing x, they validate whether the Voronoi diagram of $\mathbf{A}$ is identical to that of $\mathbf{A}'$ [LaBonte, 2021]. Then the authors used a similar method to solve the problem when $\mathbf{b} \neq 0$.

The pseudo-code of the simplified Furthest Oracle algorithm we implemented is shown in Algorithm 1. The simplified Furthest Oracle algorithm takes as input an unknown linear program characterized by constraints of the form $\mathbf{A}x \geq \mathbf{b}$, where the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are both unknown. Additionally, the algorithm is provided with a known matrix $\mathbf{A}'$ and a known vector $\mathbf{b}'$, along with a matrix D, which is an essential part of constructing the initial ellipsoid. The algorithm's output is a valid solution x that satisfies the constraint $\mathbf{A}x \geq \mathbf{b}$. If no feasible solution x is found, the algorithm will output None. The oracle plays a crucial role through the whole process. The oracle, in this context, can be perceived as a theoretical black box capable of determining whether a given inequality holds. It provides information on the furthest index of the constraints if several violations are detected. Notably, $\mathbf{A}$ and $\mathbf{b}$ may be partially or entirely unknown to us. In the context of the algorithm, the "furthest" denotes that when constraints are violated, the oracle identifies the index of the constraint with the maximum distance between the current point x and the half space represented by the violated constraint from the matrix $\mathbf{A}$ and the vector $\mathbf{b}$. This is analogous to the behavior exhibited by the Ellipsoid Method.

The iteration begins with the initial ellipsoid E every time. If the known inequality $\mathbf{A}x \geq \mathbf{b}$ holds, we could get a feasible solution x; and in the meantime, if the result of the oracle to the x is true, then x is a feasible solution to both known inequality and unknown inequality $\mathbf{A}x \geq \mathbf{b}$, the iteration will stop and the algorithm will return the x. Conversely, if the oracle returns an index i, the current ellipsoid E would be updated based on the a_i' (a_i' is the ith row of the matrix $\mathbf{A}'$) and jump into the next iteration.

Algorithm 1 The Simplified Furthest Oracle Algorithm[Bei et al., 2013]

Input: an unknown LP $\mathbf{A}x \geq \mathbf{b}$; an initial matrix $\mathbf{A}'$ and vector $\mathbf{b}'$; a symmetrical positive definite matrix D_0
Output: a valid solution x for $\mathbf{A}x \geq \mathbf{b}$, or None
Set the initial ellipsoid $E_0 = E((\mathbf{A}', \mathbf{b}'), D_0), t = 0$, and the oracle
for *each iteration* **do**
  if $\mathbf{A}'x \geq \mathbf{b}'$ *is feasible* **then**
    find a solution x and query x to the oracle
    if *the oracle returns True* **then**
      return x and **terminate**
    end
    else
      the oracle returns an index i, update the ellipsoid with a_i' and **continue**
    end
  end
  else
    constructs GenVor and GenVor$'$, run VorCheck$(\mathbf{A}', \mathbf{b}')$ and choose to **terminate** or update the ellipsoid with a_i' and **continue** based on the check results
  end
end

What if the current value of x does not satisfy the inequality $\mathbf{A}'x \geq \mathbf{b}'$? In such a scenario, we would proceed to generate two sets of Generalized Voronoi Diagrams and subsequently assess their consistency. The Voronoi diagram is a geometric partitioning of a space into regions based on the proximity to a specified set of points, termed as "sites." Each region in the Voronoi diagram is exclusively associated with a specific site, and all points within a given region are closer to the associated site than to any other site. More details about the Voronoi diagram are introduced in the Appendix G. In essence, the index returned by the oracle for a given point x in the n-dimensional space

aligns with the Voronoi cell that encompasses x and is furthest from the center, thereby facilitating a comparison between the Voronoi diagrams of matrices A and A In [Bei et al., 2013], the authors demonstrate that, given the nature of the furthest oracle, which assesses the most violated constraint, the index provided by the oracle for an input $x \in R^n$ corresponds precisely to the furthest Voronoi cell containing x. This insight is employed to verify the equivalence between the Voronoi diagram of matrix $\mathbf{A}$ and the Voronoi diagram of matrix $\mathbf{A}'$.

The underlying rationale is to subsequently evaluate the consistency between these two Voronoi diagrams derived from the matrices $\mathbf{A}$ and $\mathbf{A}'$. Based on this assessment, the algorithm proceeds with distinct outcomes. In the event of consistency, as stipulated by the Lemma 8 [Bei et al., 2013], the original algorithm undertakes the identification of some specific extreme points. Depending on this analysis, a decision is made regarding whether to terminate the process or continue the next iteration.

Conversely, in cases where the two Voronoi diagrams are not consistent, the oracle will return the furthest index, denoted as i. Subsequently, the ellipsoid is updated, and the iteration process continues, adapting to the information obtained from the furthest oracle. This dynamic process ensures that the algorithm adjusts and refines its approach based on the feedback received from the Voronoi diagrams and the furthest oracle. Details about this part can be found at Appendix G.

The difficulty in implementing the original Furthest Oracle algorithm section arises from two primary challenges. Firstly, for every Generalized Voronoi Diagram, an examination of all $(m-1)$ facets is necessary. As the value of m increases, this verification process demands a significant amount of computational time. Additionally, the complexity is compounded by the fact that the matrix $\mathbf{A}$ is unknown. Consequently, acquiring vectors x that violate the constraints $\mathbf{A}x \geq \mathbf{b}$ while simultaneously satisfying $\mathbf{A}'x \geq \mathbf{b}'$ across all dimensions becomes a challenging task Bei et al. [2013]. This dual challenge of computational intensity and the unknown nature of matrix $\mathbf{A}$ makes the implementation of this part particularly intricate.

5.2 Empirical Example

Consider the following CMDP:

$\mathcal{S} = 2, \mathcal{A} = 2, K = 1, \gamma = 0,$

$$P = \left[\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \quad \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \right], \quad r = \begin{bmatrix} 10.0 & 10.0 \\ 10.0 & 10.0 \end{bmatrix}, \quad c = \begin{bmatrix} 5.0 & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$

We construct A and b per the linear program formulation in Section 4 and approximate them as $A^{'}$ and $b^{'}$. In this example assume that both are correct approximations except for $b^{'}$ which incorrectly approximates C_1 as 4 rather than 2 in the 8^{th} index. In Appendix H We show that the feedback from the oracle is sufficient for finding a feasible solution for this CMDP

6 Conclusion and Future Work

In this work, we began with a standard constrained Markov decision process, and motivated by scenarios in which constraints are not known, extended the standard framework by considering the problem of unknown constraints. We have shown that, given a very limited set of assumptions, it is possible to find policies that respect the unknown constraints. Given a different set of assumptions that provide more information, it is possible to find not only feasible policies, but optimal policies. Furthermore, both of these outcomes can be achieved efficiently—that is, in polynomial time.

Key to the results achieved in this work are the linear-programmatic structure of constrained Markov decision processes; the ellipsoid method, which provides a way to search for a policy despite lack of knowledge of the constraints; and aspects of discrete and computational geometry, such as Voronoi diagrams and convex hulls.

Building on the work presented in this paper, in the nearer term, it would be desirable to develop complete implementations of the methods presented, and to understand the extent to which the methods presented can be applied in practical contexts. There is also space to explore varied sets of assumptions regarding the unknown constraints and the precise feedback provided, and to understand the limits of these sets of assumptions, in terms of the best policy achievable in each framework.

More broadly, we can identify four main paradigms of constrained Markov decision processes, according the nature of their constraints:

1. known, constant constraints,
2. known, changing constraints,
3. unknown, constant constraints,
4. unknown, changing constraints.

The first paradigm has been explored far more than the others. In this work, we have explored a mixture of the second and third paradigms. In future work, we suggest further exploration of the second, third and fourth paradigms. It would be desirable to learn which sets of assumptions in each paradigm allow us to obtain feasible policies, good policies, or optimal policies.

As described in Section 3, the problem of unknown constraints has been explored in the context of Bayesian optimization, but differences in frameworks between reinforcement learning and Bayesian optimization would seem to limit the applicability of techniques from Bayesian optimization to constrained Markov decision processes with unknown constraints. It would be desirable to more thoroughly explore these framework differences and the possibility of making adaptations to bridge these gaps and apply techniques from Bayesian optimization to the unknown constraint problem analyzed in this paper.

It would also be desirable to make contact with the infinite-constraint model as explored by Zhang et al. [2022], which has primarily been investigated in the first paradigm. In principle, the infinite-constraint model, combined with the possibility of unknown constraints, may be of use in robotics, because constraints in the infinite-constraint model are often of a spatial nature, and as situations vary, the constraints faced by a robot may vary, and not be entirely known.

7 Author Contributions

Jesse Schneider came up with the idea to consider unknown constraints in a constrained Markov decision process. He developed the idea, did the research, developed the motivating examples, and surveyed related work. He learned that a constrained Markov decision process is a linear program, and made the connection with the methods in [Bei et al., 2013] to find a feasible solution when the constraints are unknown. He developed the theory, the results and proofs, and analyzed the properties, strengths and shortcomings of the methods presented in this project.

Regarding the presentation on Dec. 1st, Jesse presented the slides in the Related Work and Project Contribution sections. Jesse wrote all the slides except those in Algorithm and Example, and Conclusion. (The slides in Motivation and Problem Formulation were modified in consultation with Mikhail Sinitcyn.) Jesse also participated in some discussions regarding the example demonstrated in the Algorithm and Example section of the presentation.

Regarding the final report, Jesse wrote the title, abstract, Sections 1, 2, 3, 4, 6, and Appendices A, B, C, D, E, F, and G.

Hanyang Hu interpreted and explained the algorithms we focused on in this paper (mainly Section 5) and presentation; besides, he helped Mikhail Sinitcyn to implement the examples together; he also made some revisions to the whole report.

Mikhail Sinitcyn formulated the motivating example, per [Bei et al., 2013]. and [Altman, 1999]. Collaborating with Hanyang Hu, developed the theoretical results, including the formulations for both the Constrained Markov Decision Process and the Linear Program. Created the empirical example and documented these findings in the example section of the report. Contributed substantively to section 5 and Appendix H, as well as edits throughout the document.

Regarding the presentation on December 1st, Mikhail took the lead in presenting the motivating example and the problem formulation, introducing the CMDP and furthest oracle model. Mikhail answered audience questions regarding the problem formulation and reasoning, alternate formulations, and telecommunications motivating example.

References

Eitan Altman. *Constrained Markov Decision Processes*. Stochastic Modeling. Taylor & Francis Group, 1999.

Peter F Ash and Ethan D Bolker. Recognizing Dirichlet tessellations. *Geometriae Dedicata*, 19: 175–206, 1985.

Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing Co. Pte. Ltd., 2013.

Xiaohui Bei, Ning Chen, and Shengyu Zhang. Solving Linear Programming with Constraints Unknown, 2013. arXiv version with full details: `https://arxiv.org/abs/1304.1247`.

Xiaohui Bei, Ning Chen, and Shengyu Zhang. Solving Linear Programming with Constraints Unknown. In *Automata, Languages, and Programming: 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I 42*, pages 129–142. Springer, 2015.

J Bernardo, MJ Bayarri, JO Berger, AP Dawid, D Heckerman, AFM Smith, and M West. Optimization Under Unknown Constraints. *Bayesian Statistics*, 9(9):229, 2011.

Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Number 6 in Athena Scientific Series in Optimization and Neural Computation. Athena Scientific, 1997.

Stephen Boyd and Craig Barratt. *Linear Controller Design: Limits of Performance*. Prentice-Hall (Copyright returned to authors), 1991.

Mung Chiang, Prashanth Hande, Tian Lan, Chee Wei Tan, et al. Power Control in Wireless Cellular Networks. *Foundations and Trends® in Networking*, 2(4):381–533, 2008.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. *Advances in Neural Information Processing Systems*, 30, 2017.

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, 3 edition, 2008.

Seyedshams Feyzabadi and Stefano Carpin. Risk-aware Path Planning Using Hirerachical Constrained Markov Decision Processes. In *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 297–303. IEEE, 2014.

Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

Miguel A Goberna and Marco A Lopez. Linear semi-infinite programming theory: An updated survey. *European Journal of Operational Research*, 143(2):390–405, 2002.

Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

Jacob E. Goodman, Joseph O'Rourke, and Csaba D. Tóth, editors. *Handbook of Discrete and Computational Geometry*. Discrete Mathematics and its Applications. CRC Press, 3 edition, 2018.

Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Number 2 in Algorithms and Combinatorics. Springer Berlin, Heidelberg, 2 edition, 1993.

Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, Yaodong Yang, and Alois Knoll. A Review of Safe Reinforcement Learning: Methods, Theory and Applications, 2023.

David Hartvigsen. Recognizing Voronoi Diagrams with Linear Programming. *ORSA Journal on Computing*, 4(4):369–374, 1992.

Shahin Jabbari, Ryan M Rogers, Aaron Roth, and Steven Z Wu. Learning from Rational Behavior: Predicting Solutions to Unknown Linear Programs. *Advances in Neural Information Processing Systems*, 29, 2016.

Leonid Genrikhovich Khachiyan. A polynomial algorithm in linear programming. In *Doklady Akademii Nauk*, volume 244, pages 1093–1096. Russian Academy of Sciences, 1979.

B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep Reinforcement Learning for Autonomous Driving: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

Tyler LaBonte. Finding the Needle in a High-Dimensional Haystack: Oracle Methods for Convex Optimization. 2021.

Jiří Matoušek and Bernd Gärtner. *Understanding and Using Linear Programming*. Universitext. Springer, 2007.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Douglas C. Montgomery. *Design and Analysis of Experiments*. Wiley, 7 edition, 2008.

Atsuyuki Okabe, Barry Boots, Kokichi Sugihara, and Sung Nok Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. Wiley Series in Probability and Statistics. John Wiley & Sons Ltd, 2 edition, 2000.

OpenAI. GPT-4 Technical Report, 2023.

Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction*. Texts and Monographs in Computer Science. Springer-Verlag New York Inc., 1985.

Martin L. Puterman and Timothy C. Y. Chan. *Intrduction to Markov Decision Processes*. Cambridge University Press, 2023. Pre-publication version: `https://github.com/martyput/MDP_book`.

John S Seybold. *Introduction to RF Propagation*. John Wiley & Sons, 2005.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Akifumi Wachi and Yanan Sui. Safe Reinforcement Learning in Constrained Markov Decision Processes. In *International Conference on Machine Learning*, pages 9797–9806. PMLR, 2020.

Liangyu Zhang, Yang Peng, Wenhao Yang, and Zhihua Zhang. Semi-Infinitely Constrained Markov Decision Processes. *Advances in Neural Information Processing Systems*, 35:16808–16820, 2022.

A Examples Motivating the Assumption of Limited Feedback

Cellular networks consist of, in part, pairs of transmitters and receivers. The "transmit power control" problem in a cellular network involves managing the strengths of the signals emitted by the transmitters and intended for their paired receivers. The signal emitted by a given transmitter must be strong enough so that it is received by its paired receiver, but not so strong that the signal from this transmitter interferes with other receivers [Bei et al., 2013]. The precise nature of the constraints on the emitted signals are related to "channel gains", which are often unknown in practical scenarios [Chiang et al., 2008]. We consider a reinforcement learning agent tasked with managing transmit power control in a cellular network. Due to limited resources or electronic interference, the agent only learns *which* transmitter's signal strength was set too high, without receiving any further details about the nature of the violation(s).

Other scenarios motivating the idea of limited feedback come from product design and experiments [Montgomery, 2008]. For example, when evaluating a new product with a focus group, or making a recommendation to a customer, the customer may give negative feedback without describing every

way in which the product or recommendation is undesirable. A customer may decline to purchase a recommended foodstuff because the price is too high, while failing to mention that he is on a diet and reluctant to consume highly caloric food.

B Markov Decision Processes as Linear Programs

A regular, non-constrained Markov decision process can be described as a tuple $(\mathcal{S}, \mathcal{A}, \rho, \mathcal{P}, r)$, very similarly to a constrained Markov decision process as defined in Section 2, but without the cost function c. A regular Markov decision process can be written and solved as a linear program of the form

$$\begin{aligned}
&\text{maximize } \frac{\langle \mu, r \rangle}{1-\gamma} \\
&\text{subject to } \gamma \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mathcal{P}(s' \mid s, a)\mu(s, a) + (1-\gamma)\rho(s') = \sum_{a \in \mathcal{A}} \mu(s', a) \\
&\qquad \text{for all } s' \in \mathcal{S}, \\
&\text{and } \mu(s, a) \geq 0 \text{ for all } (s, a) \in \mathcal{S} \times \mathcal{A},
\end{aligned}$$

where γ is the discount factor. A feasible solution μ to the above linear program implicitly defines a policy by the relationship

$$\pi(a \mid s) = \frac{\mu(s, a)}{\sum_{a' \in \mathcal{A}} \mu(s, a')}. \tag{1}$$

The quantity $\mu(s, a)$ can be interpreted as the total discounted joint probability that the Markov decision process is in state s and chooses action a, given certain conditions on the starting state [Puterman and Chan, 2023]. For more details on the connection between regular Markov decision processes and linear programs, consult [Puterman and Chan, 2023].

Following Altman [1999], we learn that a *constrained* Markov decision process can also be solved via a linear program of the form

$$\begin{aligned}
&\text{maximize } \frac{\langle \mu, r \rangle}{1-\gamma} \\
&\text{subject to } \gamma \sum_{s \in \mathcal{S}} \sum_{a \in \mathcal{A}} \mathcal{P}(s' \mid s, a)\mu(s, a) + (1-\gamma)\rho(s') = \sum_{a \in \mathcal{A}} \mu(s', a) \\
&\qquad \text{for all } s' \in \mathcal{S}, \\
&\text{and } \mu(s, a) \geq 0 \text{ for all } (s, a) \in \mathcal{S} \times \mathcal{A}, \\
&\text{and } \langle \mu, c_k \rangle \leq C_k \text{ for } k = 1, \ldots, K.
\end{aligned}$$

Thus, we see that the linear program for a constrained Markov decision process is simply the linear program for the corresponding unconstrained Markov decision process, augmented by the additional constraints. For details of this relationship, see [Altman, 1999].

C Applications of Related Work

This appendix extends the discussion of related work in Section 3 by describing applications of the work discussed in that section.

The textbook by Altman [1999] describes applications of constrained Markov decision processes to communications networks. Constrained Markov decision processes have also been applied to robotic path planning [Feyzabadi and Carpin, 2014], but again, the constraints are assumed known.

The safe reinforcement learning survey by Gu et al. [2023] describes many applications of constrained Markov decision processes in safe reinforcement learning, such as autonomous driving, robotics, and video compression, among others.

The work by Wachi and Sui [2020] does not contain any concrete applications of their work, but a simulated example of a Mars rover is provided.

Bayesian optimization has many applications. When constraints are known (or indeed, when there are no constraints), it has applications across science, for example in chemistry and materials science, gene and protein design, robotics, and engineering. In computer science, Bayesian optimization has applications in algorithm configuration and machine learning hyperparameter tuning [Garnett, 2023, Appendix D]. The common theme in these applications is that a function f needs to be optimized, and neither analytical techniques such as calculus nor brute-force techniques are feasible. Therefore, the points at which the function f should be evaluated must be carefully chosen.

When constraints are unknown, we have found one motivating example and one application of Bayesian optimization. The motivating example concerns a business attempting to maximize revenue, subject to unknown constraints on customer response [Garnett, 2023]. Unfortunately, this example is not described in detail. The application concerns a large simulation of the effects of changes in health care policy in the United States on the health insurance purchasing behavior of the public [Bernardo et al., 2011]. In the simulation, the response of the public to health care policy changes is subject to unknown constraints concerning the public's response elasticity (roughly speaking, the change in the public's health insurance purchasing behavior as aspects of health care policy are changed). The elasticities "can only be found by running the simulator" [Bernardo et al., 2011], which means they are unknown in advance.

The work by Zhang et al. [2022] does not contain any concrete applications of their work, but they present a simulated sewage discharge example in which a finite number of sewage discharge points in $[0, 1]^2 \subset \mathbb{R}^2$ must be managed so that points $x \in [0, 1]^2 \subset \mathbb{R}^2$ are kept below their constraints.

D The Ellipsoid Method

The ellipsoid method is a convex optimization technique that was very important theoretically when it was introduced in the 1970s, because it was used by Khachiyan to prove that linear programs can be solved in polynomial time, in the worst case [Khachiyan, 1979]. However, in practice, it is typically slower than the simplex method for solving linear programs [Grötschel et al., 1993, Bertsimas and Tsitsiklis, 1997].

There are many descriptions of the ellipsoid method available [Boyd and Barratt, 1991, Grötschel et al., 1993, Matoušek and Gärtner, 2007, Bertsimas and Tsitsiklis, 1997]. Here, we provide a non-exhaustive introduction to the basic ellipsoid method, following [Bertsimas and Tsitsiklis, 1997]; more details can be found therein, or in the aforementioned references. See Figure 1 for an illustration of the following description.

Given a convex set $P \coloneqq \{\mathbf{x} \in \mathbb{R}^n \colon \mathbf{A}\mathbf{x} \geq \mathbf{b}\}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^n$, we seek a point in P. At every time step t, we will propose a point $\mathbf{x}_t \in \mathbb{R}^n$ to an oracle, which will give feedback regarding whether our proposed point is in P.

At time $t = 0$, we begin with an initial ellipsoid E_0 containing P. We construct a sequence of ellipsoids E_t with center points $\mathbf{x}_t$, such that $P \subset E_t$ for all t. At each time step, we propose to the oracle the point $\mathbf{x}_t$. If $\mathbf{x}_t \in P$, then the algorithm terminates. If $\mathbf{x}_t \notin P$, then we receive feedback about a constraint $\mathbf{a}'\mathbf{x} \geq b$ that $\mathbf{x}_t$ violates, where $\mathbf{x} \in P$, and $\mathbf{a}'$ is a row of $\mathbf{A}$, and b is the corresponding entry of $\mathbf{b}$. Because $\mathbf{x}_t$ violates this constraint, we know that $\mathbf{a}'\mathbf{x}_t < b$.

Since this implies that $\mathbf{a}'\mathbf{x} \geq \mathbf{a}'\mathbf{x}_t$, we know that P is in the half-space defined by $\mathbf{a}'\mathbf{x} \geq \mathbf{a}'\mathbf{x}_t$, *i.e.*, $P \subset \left\{\mathbf{x} \in \mathbb{R}^N \colon \mathbf{a}'\mathbf{x} \geq \mathbf{a}'\mathbf{x}_t\right\}$. We therefore define the next ellipsoid E_{t+1} to be the smallest ellipsoid containing $E_t \cap \left\{\mathbf{x} \in \mathbb{R}^N \colon \mathbf{a}'\mathbf{x} \geq \mathbf{a}'\mathbf{x}_t\right\}$. (The formula for E_{t+1} can be derived analytically; consult [Bertsimas and Tsitsiklis, 1997] or any of the other aforementioned references for precise mathematical details.) We then continue by querying the oracle regarding the center point $\mathbf{x}_{t+1}$ of the new ellipsoid E_{t+1}.

A key reason this procedure can find a point in P is because it can be shown that the volumes of the succeeding ellipsoids decrease rapidly, so the procedure either finds a point in P, declares that P was

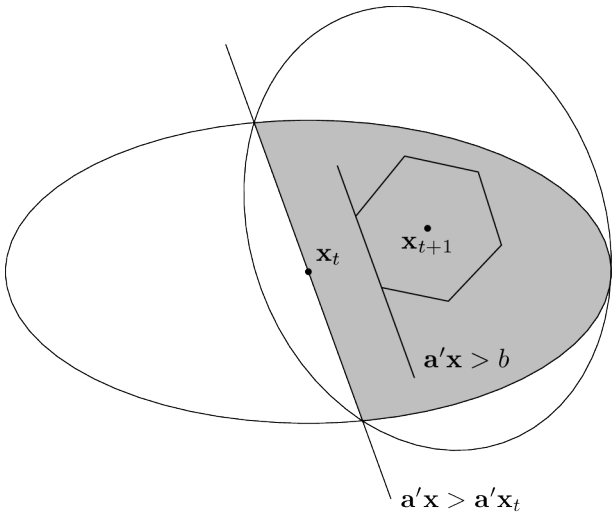

Figure 1: This figure illustrates one iteration of the basic ellipsoid method, in which $\mathbf{x}_t \notin P$, but $\mathbf{x}_{t+1} \in P$. The shaded region is the intersection $E_t \cap \left\{\mathbf{x} \in \mathbb{R}^N \colon \mathbf{a}'\mathbf{x} \geq \mathbf{a}'\mathbf{x}_t\right\}$, and the new ellipsoid E_{t+1} is the smallest ellipsoid containing the shaded region.

empty to begin with. Note that in dimension $n = 1$, the ellipsoid method is very similar to binary search.

E Technical Definitions

In this appendix, we define terms used in Section 4 concerning our main result, Proposition 1.

The first term we define is the input size of a constrained Markov decision process. This definition comes from conventional definition of the input size of a linear program.

Definition 1. *The input size L of a constrained Markov decision process is its binary size, as a linear program. It is of order $O\left(|\mathcal{S}|^2 |\mathcal{A}| + |\mathcal{S}| |\mathcal{A}| K\right)$, omitting a logarithmic factor for clarity.*

In linear programming, input size is, roughly speaking, the number of bits needed to write down the input to a linear programming algorithm [Matoušek and Gärtner, 2007].

The authors of the paper [Bei et al., 2013] define the *input size* L to be an upper bound on the binary size of the linear program, since they are concerned with a linear program whose constraints they do not know. For a linear program with m constraints, n variables, and maximum entry N, $L = O\left(mn \log(N)\right)$. It is from this definition that we derive Definition 1. To see this, note that a linear program for a regular Markov decision process involves a matrix of dimensions $|\mathcal{S}| \times (|\mathcal{S}| \cdot |\mathcal{A}|)$. With K additional constraints added, the matrix would be $(|\mathcal{S}| + K) \times (|\mathcal{S}| \cdot |\mathcal{A}|)$. Therefore its input size would be $O\left((|\mathcal{S}| + K)\left(|\mathcal{S}| \cdot |\mathcal{A}|\right) \log(N)\right)$.

In the statement of Proposition 1, we have simplified this expression, and omitted the logarithmic factor in order to bring into focus the most relevant aspects of our proposition, from the perspective of constrained Markov decision processes.

The next definition concerns the non-degeneracy referenced in Theorem 1 from [Bei et al., 2013].

Definition 2. *An $m \times n$ matrix A is non-degenerate if at most n rows of A (when scaled to unit length) are equidistant to any point p on the unit sphere in $\mathbb{R}^n$.*

This is a technical condition required in [Bei et al., 2013], which should not occur in practice. Even if it were to occur, a small perturbation can ensure that this condition is satisfied [Bei et al., 2013].

F Alternate Framework

In this appendix, we present an alternate version of the unknown constraints problem in which it is possible to not only find a feasible policy, but an optimal policy.

F.1 Introduction and Motivation

As described in Section 1 and Appendix A, there are scenarios involving transmitters and receivers in a cellular network in which the signal strength constraints are unknown. As in Appendix A, we consider a reinforcement learning agent tasked with managing transmit power control, subject to unknown signal strength constraints. In order to achieve stronger results than were achieved in Section 4, we will assume that additional information is known. This will be described in Section F.3. A new facet of this framework is that the agent faces a known, varying constraint, which we may think of as being due to changes in environmental conditions. It is known that the propagation of Ultra High Frequency radio waves, which are used for cellular signals, is affected by atmospheric conditions [Seybold, 2005]. The agent will attempt to manage the signal strengths, and it will be regularly re-calibrated by an outside entity, which will reveal the optimal signal strengths that the agent should have set.

F.2 Related Work

The survey in Section 3 is still relevant, but we note that for this framework, we rely on certain results presented in [Jabbari et al., 2016].

F.3 Problem Formulation

As in Section 2, we again consider a constrained Markov decision process, with the same objective. We assume that $\mathcal{S}$, $\mathcal{A}$, and the reward function r are known, and that c_K is known. We also assume that the final constraint C_K is drawn independently and identically from the same distribution $\mathscr{D}$ at each iteration t, so at time t we may write the final constraint as $C_K^{(t)}$. We assume that $C_K^{(t)}$ is known at each time t. No other components of the constrained Markov decision process are assumed known. At each time t, we seek an optimal policy for our constrained Markov decision process.

As before, in order to make progress, we make further assumptions connected with the notion of limited feedback. At each iteration t, we propose a solution $\hat{\mu}^{(t)}$, and we then observe the true optimal solution $\mu^{(t)}$. We define a mistake at time t to be the event that $\hat{\mu}^{(t)} \neq \mu^{(t)}$.

Note that, in order to make the problem non-trivial, it is necessary to allow the known constraint C_K to vary at each iteration. If the known constraint were not allowed to vary, then all of the constraints would be constant, and therefore the optimal solution would be constant. The first iteration would therefore reveal the optimal solution for *every* iteration, thereby rendering the problem trivial.

The techniques applied to this problem formulation will again rely on the fact that the optimal solution to a constrained Markov decision process can be found via a linear program. Because of this, we now make some further assumptions regarding the linear program that corresponds to the constrained Markov decision process.

The linear program corresponding to the constrained Markov decision process, *excluding* c_K and C_K, involves an unknown, constant set of constraints. We denote the convex polytope defined by these constraints as $\mathscr{P} \subset \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$. In iteration t, the new constraint $C_K^{(t)}$ is revealed, which defines an additional linear program constraint $\mathscr{N}^{(t)} := \left\{\mu \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \colon \langle \mu, c_K \rangle \leq C_K^{(t)}\right\}$. At time t, the total set of linear program constraints can therefore be written as $\mathscr{P}^{(t)} := \mathscr{P} \cap \mathscr{N}^{(t)}$.

First, we assume that at each iteration t, our problem has a unique solution.

Assumption 1. *The optimal solution to the linear program corresponding to the constrained Markov decision process at time t is unique for all t.*

Next, we make an assumption that holds without loss of generality, up to scaling.

Assumption 2. *For each row p of the unknown, constant polytope $\mathscr{P}$, $\|p\|_\infty \leq 1$.*

For the next two assumptions, we use generic notation for simplicity, so that the set $\mathscr{P}$ involves a constraint matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$. These assumptions are made to ensure non-degeneracy.

Assumption 3. *Any $d-1$ rows of the matrix $\mathbf{A}$ have rank $d-1$.*

Assumption 4. *Each vertex of $\mathscr{P}^{(t)}$ is the intersection of exactly d hyperplanes of $\mathscr{P}^{(t)}$.*

F.3.1 The Edges of the Polytope $\mathscr{P}$

The methods applied in Section F.4 depend on the set of edges of the polytope $\mathscr{P}$, which we denote $E_{\mathscr{P}}$. In particular, they depend on the number of edges of the polytope, which we denote $|E_{\mathscr{P}}|$. Because of the importance of the cardinality of the set of edges $E_{\mathscr{P}}$ in what follows, we attempt to illuminate its relation to our constrained Markov decision process.

First, we bound $|E_{\mathscr{P}}|$ for a general polytope $\mathscr{P}$, meaning that $\mathscr{P} = \left\{\mathbf{x} \in \mathbb{R}^d \colon \mathbf{A}\mathbf{x} \leq \mathbf{b}\right\}$, where $\mathbf{A} \in \mathbb{R}^{m \times d}$ and $\mathbf{b} \in \mathbb{R}^m$.

The cyclic d-polytope with n vertices, $C_d(n)$, is a polytope in $\mathbb{R}^d$ whose importance for our purposes is that it has the greatest number of edges of any convex polytope with n vertices [Goodman et al., 2018]. In particular, by Theorem 17.3.3 and 17.3.4 of [Goodman et al., 2018], letting $\delta = d - 2\lfloor d/2 \rfloor$, we have

$$|E_{\mathscr{P}}| \leq \frac{n - \delta(n-3)}{n-2} \sum_{j=0}^{\lfloor d/2 \rfloor} \binom{n-1-j}{2-j} \binom{n-2}{2(j-1)+\delta}, \tag{2}$$

where the right-hand side is the number of edges of $C_d(n)$. For more information, consult [Goodman et al., 2018, Section 17.3].

We now relate the above bound on the number of edges of $\mathscr{P}$ to the constrained Markov decision process context. In Appendix E, it was mentioned that the matrix associated with a constrained Markov decision process with K constraints has dimensions $(|\mathcal{S}| + K) \times (|\mathcal{S}| \cdot |\mathcal{A}|)$. The number of vertices (denoted n in (2)) and the dimension (denoted d in (2)) of the associated polytope will both grow as $|\mathcal{S}|$, $|\mathcal{A}|$ and K grow. Therefore, it is clear from (2) that the upper bound on number of edges of the polytope associated with the constrained Markov decision process will grow as the state space, action space and number of constraints grow.

F.4 Theoretical Results and Discussion

F.4.1 Theoretical Results

As in Section 4, we begin by considering a constrained Markov decision process as a linear program. In this framework, our methods take advantage of the fact that at each iteration, we receive more valuable information than in the framework laid out in Section 2. Specifically, we observe the new value of the final constraint $C_K^{(t)}$, and the optimal solution $\mu^{(t)}$. The essential idea of the methods we apply is to keep track of the optimal solutions observed up to time t, *i.e.*, $\left\{\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(t-1)}\right\}$, and use them, along with the observed constraint $C_K^{(t)}$ and knowledge of the objective function, to make the prediction $\hat{\mu}^{(t)}$.

The method we apply is a slight adaptation of the method termed `LearnHull` in [Jabbari et al., 2016]. We will call our version of `LearnHull` by the name `LearnPolicy`. `LearnPolicy` computes the convex hull of the set of observed optimal solutions up to time t, $\left\{\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(t-1)}\right\}$. Denote the convex hull of the points $\left\{\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(t-1)}\right\}$ by $\mathscr{C}^{(t-1)}$. At iteration t, the method `LearnPolicy` uses the observation $C_K^{(t)}$ to determine the half-space $\mathscr{N}^{(t)}$, and its computation of $\mathscr{C}^{(t-1)}$ to maximize the objective function over the region $\mathscr{C}^{(t-1)} \cap \mathscr{N}^{(t)}$.

The algorithm `LearnPolicy` is presented formally in Algorithm 2.

Algorithm 2 `LearnPolicy`

Input: State space $\mathcal{S}$, action space $\mathcal{A}$, reward function r, cost c_K
Initialize: $\mathscr{C}^{(0)} = \emptyset$
for $t = 1, \ldots, T$ **do**
  Observe $C_K^{(t)} \sim \mathscr{D}$
  Set $\mathscr{N}^{(t)} = \left\{\mu \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \colon \langle \mu, c_K \rangle \leq C_K^{(t)}\right\}$
  Predict $\hat{\mu}^{(t)} = \arg\max_{\mu \in \mathscr{C}^{(t-1)} \cap \mathscr{N}^{(t)}} \langle \mu, r \rangle / (1-\gamma)$
  Compute policy $\hat{\pi}^{(t)}$ according to $\hat{\pi}^{(t)}(a \mid s) = \frac{\hat{\mu}^{(t)}(s,a)}{\sum_{a' \in \mathcal{A}} \hat{\mu}^{(t)}(s,a')}$ for $(s,a) \in \mathcal{S} \times \mathcal{A}$
  Observe $\mu^{(t)} = \arg\max_{\mu \in \mathscr{P}^{(t)}} \langle \mu, r \rangle / (1-\gamma)$
  if $\hat{\mu}^{(t)} \neq \mu^{(t)}$ **then**
    $\mathscr{C}^{(t)} =$ Convex hull of $\mathscr{C}^{(t-1)} \cup \left\{\mu^{(t)}\right\}$
  else
    $\mathscr{C}^{(t)} = \mathscr{C}^{(t-1)}$
  end if
end for

It is important to note that every prediction $\hat{\mu}^{(t)}$ made by `LearnPolicy` is a feasible solution to the linear program, and therefore implicitly defines a predicted policy $\hat{\pi}^{(t)}$ that satisfies the constrained Markov decision process. In other words, `LearnPolicy` does not make mistakes by predicting infeasible policies, but only by predicting sub-optimal policies. We state this important property of `LearnPolicy` as a proposition.

Proposition 2. *At time t, the algorithm `LearnPolicy` makes a prediction $\hat{\mu}^{(t)}$ that is feasible, i.e., $\hat{\mu}^{(t)} \in \mathscr{P}^{(t)}$.*

Proof. First note that we have $\hat{\mu}^{(t)} \in \mathscr{C}^{(t-1)} \cap \mathscr{N}^{(t)}$. By definition, any point from this set will satisfy the new, known constraint $\mathscr{N}^{(t)}$, so $\hat{\mu}^{(t)} \in \mathscr{N}^{(t)}$.

It only remains to show that $\hat{\mu}^{(t)} \in \mathscr{P}$. Because $\hat{\mu}^{(t)} \in \mathscr{C}^{(t-1)}$, $\hat{\mu}^{(t)}$ must be a convex combination of $\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(t-1)}$, so

$$\hat{\mu}^{(t)} = \sum_{i=1}^{t-1} \alpha_i \mu^{(i)}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_{t-1} \geq 0$ and $\sum_{i=1}^{t-1} \alpha_i = 1$. But note that because $\mu^{(1)}, \mu^{(2)}, \ldots, \mu^{(t-1)} \in \mathscr{P}$, and $\mathscr{P}$ is convex, $\mathscr{P}$ must contain $\sum_{i=1}^{t-1} \alpha_i \mu^{(i)} = \hat{\mu}^{(t)}$, so $\hat{\mu}^{(t)} \in \mathscr{P}$.

Thus, we have shown that $\hat{\mu}^{(t)} \in \mathscr{P} \cap \mathscr{N}^{(t)} = \mathscr{P}^{(t)}$, as desired. □

We now state and prove our main results for this framework, which are direct applications of results concerning `LearnHull` from [Jabbari et al., 2016].

Proposition 3. *For $T > 0$ and any constraint distribution $\mathscr{D}$, the expected number of mistakes of `LearnPolicy` after T iterations is bounded by $O\left(|E_{\mathscr{P}}| \log(T)\right)$.*

Proof. Apply Theorem 9 from [Jabbari et al., 2016]. □

If it is possible to run many independent instances of `LearnPolicy` and aggregate their individual predictions, then the probability of making a mistake after T iterations can be bounded.

Proposition 4. *Let $\delta \in (0, 1/2)$. Suppose $\lceil 18 \log(1/\delta) \rceil$ independent instances of `LearnPolicy` are run, using independently drawn samples, and the mode of the individual predictions is used as the overall prediction. Suppose also that the independent instances are run for $T = O\left(|E_{\mathscr{P}}| \log(1/\delta)\right)$ iterations. Then the probability that the modal prediction is incorrect on day $T+1$ is at most δ.*

Proof. Apply Theorem 11 from [Jabbari et al., 2016]. □

F.4.2 Discussion

We now present some analysis of the methods and results presented in Section F.4.1. First, as we saw in Proposition 3, the expected number of mistakes made by `LearnPolicy` is bounded by an expression involving $|E_{\mathscr{P}}|$. However, we saw in (2) that the bound on $|E_{\mathscr{P}}|$ will grow as the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the number of constraints K grow. Therefore, we can expect that `LearnPolicy` will perform worse on larger constrained Markov decision processes with more constraints.

Second, it is noted in [Jabbari et al., 2016] that, if all coefficients of the unknown constraints for $\mathscr{P}$ can be represented with N bits, then the algorithm `LearnHull` can be implemented in $\text{poly}(T, N, d)$ time, where d comes from the fact that `LearnHull` attempts to predict optimal solutions in $\mathbb{R}^d$. In the context of constrained Markov decision processes, we seek optimal solutions in $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$, so we would expect that, if we make the same assumption regarding N, then `LearnPolicy` can be implemented in $\text{poly}\left(T, N, |\mathcal{S}|, |\mathcal{A}|\right)$ time.

In general, according to the Upper Bound Theorem, computing the convex hull of t points in $|\mathcal{S}||\mathcal{A}|$-dimensional space is $\Theta\left(t^{\lfloor |\mathcal{S}||\mathcal{A}| \rfloor/2}\right)$ [de Berg et al., 2008]. The *online* computation of a convex hull, *i.e.*, with points arriving one-by-one, can be $\Theta(\log t)$, *in* $\mathbb{R}^2$ [Preparata and Shamos, 1985]; it is not clear if there are fast methods for online computation of a convex hull in arbitrary dimension. It seems that, in order to achieve the stated polynomial time result in [Jabbari et al., 2016], the online computation of the convex hull must be done faster than $\Theta\left(t^{\lfloor |\mathcal{S}||\mathcal{A}| \rfloor/2}\right)$.

The prediction of $\hat{\mu}^{(t)}$ over $\mathscr{C}^{(t-1)} \cap \mathscr{N}^{(t)}$ can be done via linear programming, and thus in polynomial time. This appears to be the main computational bottleneck in `LearnPolicy` (and therefore `LearnHull`).

In Section F.3, we assumed that c_K is known. However, `LearnPolicy` generalizes to the case when c_K and C_K are together drawn independently from a fixed, unknown distribution $\mathscr{D}$ over $\mathbb{R}^{|\mathcal{S}||\mathcal{A}|} \times \mathbb{R}$, since the methods applied from [Jabbari et al., 2016] are also applicable to this case. Propositions 2, 3 and 4 still directly apply to this more general case.

In Proposition 4, we supposed that many independent instances of `LearnPolicy` could be run. This does not appear to be applicable to our motivating example of a reinforcement agent managing signal strengths in a cellular network. However, it would be applicable in simulations.

It is important to note that the growth in the expected number of mistakes made by `LearnPolicy` is only logarithmic in T. This implies that, as T becomes large, `LearnPolicy` should eventually rarely make mistakes, and therefore its performance can be expected to improve as it is run for more iterations, at least asymptotically.

F.5 Comparison of Original and Alternate Frameworks

Having presented and analyzed the alternate framework, we now compare the two frameworks and the results they achieve.

In the original framework, analyzed in Section 4, we receive very limited feedback to our proposals, and we are able to find a feasible policy for our constrained Markov decision process in polynomial time. We have no assurance that we can find a good, much less an optimal, policy, only that we can eventually find a policy that satisfies the constrained Markov decision process. As we saw in Section 4.2, it is very difficult, within the original framework, to strengthen the result in any way. In the alternate framework, analyzed in Section F.4, we have the advantage of learning, after each proposal, the optimal solution. This, along with knowledge of the (changing) constraint, is key to forming the convex hulls $\mathscr{C}^{(t-1)}$ and ensuring that each proposal $\hat{\mu}^{(t)}$ is feasible. The fact that the expected number of mistakes made by `LearnPolicy` is logarithmic in T implies that `LearnPolicy` will, eventually, rarely make mistakes as T becomes large. We have no comparable result in the original framework. In the alternate framework, we are even able to find an optimal solution with high probability, if we can run enough copies of `LearnPolicy` for long enough.

The main similarities of the results for both frameworks are that both can run in polynomial time, although the details are somewhat different in each case. In both frameworks, performance will become worse as the state space $\mathcal{S}$, the action space $\mathcal{A}$, and the number of constraints K grow.

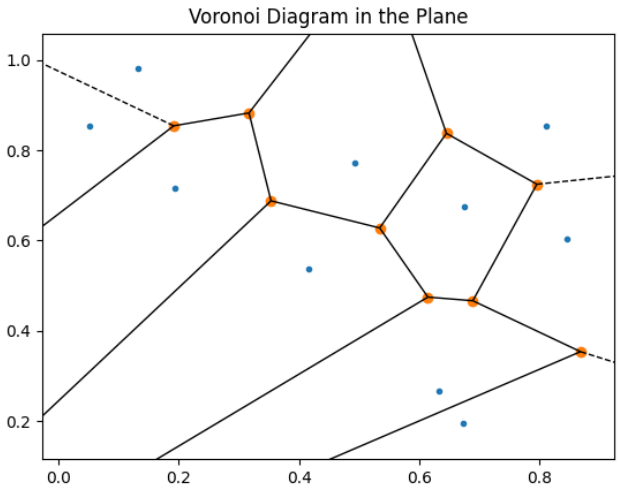

Figure 2: This figure shows (part of) the Voronoi diagram of a randomly generated set of ten points in the plane.

Furthermore, both methods can find policies that satisfy a constrained Markov decision process, although as we have seen, in the alternate framework we can do better than in the original framework. In sum, by making a different, but stronger set of assumptions, we have shown we can find feasible and even optimal policies to a constrained Markov decision process with (mostly) unknown constraints.

G Voronoi Diagrams

In this appendix, we present a brief introduction to Voronoi diagrams, and their use in finding feasible points for linear programs with unknown constraints. We assume no prior knowledge of Voronoi diagrams. In the introductory section, we provide a basic definition of a Voronoi diagram, and describe some of their applications. In the subsequent section, we describe how Voronoi diagrams arise and are used in the work by Bei et al. [2013], in order to find feasible points for linear programs with unknown constraints.

G.1 Introduction

First, we define a basic form of a Voronoi diagram.

Definition 3. *Let $P \coloneqq \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of distinct points in the plane, equipped with the Euclidean metric d. Let $I_n = \{1, 2, \ldots, n\}$. We define the Voronoi polygon, or Voronoi cell, associated with p_i to be the set*

$$\mathsf{Vor}(i) \coloneqq \left\{p \in \mathbb{R}^2 \colon d(p, p_i) \leq d(p, p_j) \text{ for } j \in I_n \setminus \{i\}\right\},$$

and the set

$$\mathsf{Vor}(P) \coloneqq \{\mathsf{Vor}(1), \mathsf{Vor}(2), \ldots, \mathsf{Vor}(n)\}$$

to be the Voronoi diagram generated by P.

We abbreviate $\mathsf{Vor}(P)$ by Vor when the context is clear. See Figure 2 for an example of a Voronoi diagram.

Voronoi diagrams have found applications in "anthropology, archaeology, astronomy, biology, cartography, chemistry, computational geometry, crystallography, ecology, forestry, geography, geology, linguistics, marketing, metallography, meteorology, operations research, physics, physiology, remote sensing, statistics, and urban and regional planning", among other disciplines [Okabe et al., 2000], We briefly describe an application to social geography, loosely based on material in [de Berg et al., 2008]. Suppose in Town X, there are n supermarkets $s_1, s_2, \ldots, s_n$, and the population of Town

X is indifferent to the shopping experience at any of the supermarkets, so everyone shops at the nearest supermarket. This model of behavior implicitly divides Town X into Voronoi polygons of the supermarkets, so that the supermarkets generate a Voronoi diagram of Town X.

More generally, the common theme in the application of Voronoi diagrams is that is there exists a *space*, a set of *sites* p in the space, and each site p exerts some level of influence on every point x in the space. Then the *region* associated with the site p is the set of all points x in the space for which the influence of p is the strongest [Aurenhammer et al., 2013].

Voronoi diagrams can be defined in spaces more abstract than the plane, for example $\mathbb{R}^n$, or potentially in any space with a notion of distance, *i.e.*, a metric space. Particularly relevant for the purposes of this appendix, Voronoi diagrams can be defined on the n-dimensional unit sphere; this will be described in Section G.2.

G.2 Voronoi Diagrams in the UnknownLP Problem

G.2.1 Background

We begin with a few preliminaries. First, we use notation similar to that used by Bei et al. [2013], because using constrained Markov decision process notation obscures the essential working of the methods developed by the same authors.

We consider a system of linear inequalities in the form $\mathbf{A}\mathbf{x} \geq \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{b} \in \mathbb{R}^m$. We denote the m rows of $\mathbf{A}$ by $\mathbf{a}_1, \ldots, \mathbf{a}_m \in \mathbb{R}^n$, and the components of $\mathbf{b}$ by $b_1, \ldots, b_m$. We define the unit sphere in $\mathbb{R}^n$ to be the set

$$S^{n-1} \coloneqq \left\{ (p_1, \ldots, p_n) \in \mathbb{R}^n \colon \sqrt{p_1^2 + \cdots + p_n^2} = 1 \right\},$$

and we will in general use the notation $\|\cdot\|$ to denote the standard Euclidean norm, and the notation $\langle \cdot, \cdot \rangle$ to denote the standard inner product.

As described in Section 4, the essential idea of the methods developed by Bei et al. [2013] is to apply the ellipsoid method to search for $(\mathbf{A}, \mathbf{b})$, so that at time t, the center of the current ellipsoid is $(\mathbf{A}_t, \mathbf{b}_t)$. A feasible point $\mathbf{x}$ for a linear program with constraints $\mathbf{A}_t\mathbf{x} \geq \mathbf{b}_t$ is then proposed to the oracle as a possible feasible point for the linear program with constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$. If $\mathbf{x}$ satisfies $(\mathbf{A}, \mathbf{b})$, then the method terminates, otherwise the oracle returns information about a separating hyperplane, which can be used to construct the next ellipsoid at time $t + 1$. Complications arise, however, when the constraints defined by $(\mathbf{A}_t, \mathbf{b}_t)$ cannot be satisfied by any point. In this case, a separating hyperplane is still needed in order to continue the ellipsoid method.

We note that, in the furthest oracle model, the oracle returns the index $i \in \{1, \ldots, m\}$ of the constraint that a proposed solution $\mathbf{x}$ violates the most, as measured by Euclidean distance. If $\mathbf{x}$ violates the constraint $\langle \mathbf{a}_i, \mathbf{x} \rangle \geq b_i$, then $\langle \mathbf{a}_i, \mathbf{x} \rangle < b_i$, and the distance between $\mathbf{x}$ and the half-space $\{\mathbf{z} \in \mathbb{R}^n \colon \langle \mathbf{a}_i, \mathbf{z} \rangle \geq b_i\}$ is given by

$$\frac{b_i - \langle \mathbf{a}_i, \mathbf{x} \rangle}{\|\mathbf{a}_i\|}. \tag{3}$$

We first consider the homogeneous case, *i.e.*, $\mathbf{b} = \mathbf{0}$, and then generalize to the inhomogeneous case.

G.2.2 Homogeneous Case

We consider a linear program with constraints $\mathbf{A}\mathbf{x} \geq \mathbf{0}$. We assume, without loss of generality, that $\|\mathbf{a}_1\| = \cdots = \|\mathbf{a}_m\| = 1$, so that the m rows of $\mathbf{A}$ lie on S^{n-1}. Furthermore, we can propose points $\mathbf{x}$ such that $\|\mathbf{x}\| = 1$.

Considering the distance given in (3) in the homogeneous case, we see that this distance is maximized when $-\langle \mathbf{a}_i, \mathbf{x} \rangle$ is maximized, so the oracle will return the index of the violated constraint i such that $-\langle \mathbf{a}_i, \mathbf{x} \rangle$ is maximized.

Given a proposed point $\mathbf{x}$, we will also be led to consider the point $\mathbf{y} \coloneqq -\mathbf{x}$, for reasons to be explained shortly. Thus, among the indices i such that $\langle \mathbf{a}_i, \mathbf{y} \rangle > 0$, the oracle returns the index i such

that $\langle \mathbf{a}_i, \mathbf{y} \rangle$ is maximized. (Since $\mathbf{x} \in S^{n-1}$, we may visualize the point $\mathbf{y} \in S^{n-1}$ as antipodal to $\mathbf{x}$ on the unit sphere. For example, if $n = 3$, and $\mathbf{x} = (0, 0, 1)$ lies at the north pole, then $\mathbf{y} = (0, 0, -1)$ lies at the south pole.)

We now describe how the feedback provided by the furthest oracle, in response to a proposed point $\mathbf{x}$, implicitly provides information about the Voronoi diagram generated by $\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}$ on the sphere S^{n-1}. Note that, for any point $\mathbf{z} \in S^{n-1}$ and $i, j \in \{1, \ldots, m\}$,

$$\begin{aligned}
&\langle \mathbf{a}_i, \mathbf{z} \rangle \geq \langle \mathbf{a}_j, \mathbf{z} \rangle \\
\iff &-\langle \mathbf{a}_i, \mathbf{z} \rangle \leq -\langle \mathbf{a}_j, \mathbf{z} \rangle \\
\iff &\langle \mathbf{z}, \mathbf{z} \rangle - 2\langle \mathbf{a}_i, \mathbf{z} \rangle + \langle \mathbf{a}_i, \mathbf{a}_i \rangle \leq \langle \mathbf{z}, \mathbf{z} \rangle - 2\langle \mathbf{a}_j, \mathbf{z} \rangle + \langle \mathbf{a}_j, \mathbf{a}_j \rangle \\
\iff &\langle \mathbf{z} - \mathbf{a}_i, \mathbf{z} - \mathbf{a}_i \rangle \leq \langle \mathbf{z} - \mathbf{a}_j, \mathbf{z} - \mathbf{a}_j \rangle \\
\iff &\|\mathbf{a}_i - \mathbf{z}\|^2 \leq \|\mathbf{a}_j - \mathbf{z}\|^2 \\
\iff &\|\mathbf{a}_i - \mathbf{z}\| \leq \|\mathbf{a}_j - \mathbf{z}\|.
\end{aligned}$$

Thus, when an (infeasible) point $\mathbf{x}$ is proposed to the oracle, the oracle returns the index i of a violated constraint such that $\langle \mathbf{a}_i, \mathbf{y} \rangle$ is maximized, which is equivalent to the condition that $\mathbf{y}$ is closer to $\mathbf{a}_i$ that it is to any of the other rows of $\mathbf{A}$. If we define

$$\begin{aligned}
\mathsf{Vor}(i) &:= \left\{ \mathbf{z} \in S^{n-1} \colon \langle \mathbf{a}_i, \mathbf{z} \rangle \geq \langle \mathbf{a}_j, \mathbf{z} \rangle, j = 1, \ldots, m \right\} \\
&= \left\{ \mathbf{z} \in S^{n-1} \colon \|\mathbf{a}_i - \mathbf{z}\| \leq \|\mathbf{a}_j - \mathbf{z}\|, j = 1, \ldots, m \right\},
\end{aligned}$$

and $\mathsf{Vor} = \mathsf{Vor}\left(\{\mathbf{a}_1, \ldots, \mathbf{a}_m\}\right)$, then we see that proposals $\mathbf{x}$ provide information about the Voronoi diagram generated by the rows of $\mathbf{A}$ on the sphere. In particular, if the oracle returns i in response to the proposal $\mathbf{x}$, then we deduce that $\mathbf{y} \in \mathsf{Vor}(i)$.

We are interested in comparing the Voronoi diagrams generated by the rows of $\mathbf{A}$ and $\mathbf{A}_t$ in order to find the desired separating hyperplane. We may denote the rows of $\mathbf{A}_t$ as $\mathbf{a}'_1, \ldots, \mathbf{a}'_m$. (The particular value of t is unimportant for this appendix, so the prime symbol suffices.) We note that, in the course of applying the ellipsoid method, there is no guarantee that any of $\mathbf{a}'_1, \ldots, \mathbf{a}'_m \in S^{n-1}$, which introduces a complication that must be addressed. This is accomplished by use of a *weighted* Voronoi diagram, which we will now describe.

As noted, the points $\mathbf{a}'_1, \ldots, \mathbf{a}'_m$ do not necessarily lie on S^{n-1}. However, via scaling, the points $\mathbf{a}'_1/\|\mathbf{a}'_1\|, \ldots, \mathbf{a}'_m/\|\mathbf{a}'_m\|$ do lie on the unit sphere. The *unweighted* Voronoi diagram generated by $\mathbf{a}'_1/\|\mathbf{a}'_1\|, \ldots, \mathbf{a}'_m/\|\mathbf{a}'_m\|$ on S^{n-1} would have cells of the form

$$\begin{aligned}
\mathsf{Vor}'_{\mathrm{UW}}(i) &:= \left\{ \mathbf{z} \in S^{n-1} \colon \left\| \frac{\mathbf{a}'_i}{\|\mathbf{a}'_i\|} - \mathbf{z} \right\| \leq \left\| \frac{\mathbf{a}'_j}{\|\mathbf{a}'_j\|} - \mathbf{z} \right\|, j = 1, \ldots, m \right\} \\
&= \left\{ \mathbf{z} \in S^{n-1} \colon \left\langle \frac{\mathbf{a}'_i}{\|\mathbf{a}'_i\|}, \mathbf{z} \right\rangle \geq \left\langle \frac{\mathbf{a}'_j}{\|\mathbf{a}'_j\|}, \mathbf{z} \right\rangle, j = 1, \ldots, m \right\}.
\end{aligned}$$

However, we are interested in finding a separating hyperplane between $\mathbf{A}$ and $\mathbf{A}_t$, *not* between $\mathbf{A}$ and the latter with rescaled rows that lie on the unit sphere. This leads us to consider instead the *weighted* Voronoi diagram generated by $\mathbf{a}'_1/\|\mathbf{a}'_1\|, \ldots, \mathbf{a}'_m/\|\mathbf{a}'_m\|$, which has cells of the form

$$\mathsf{Vor}'_{\mathrm{W}}(i) := \left\{ \mathbf{z} \in S^{n-1} \colon \langle \mathbf{a}'_i, \mathbf{z} \rangle \geq \left\langle \mathbf{a}'_j, \mathbf{z} \right\rangle, j = 1, \ldots, m \right\}.$$

In the weighted Voronoi diagram, each point $\mathbf{a}'_i/\|\mathbf{a}'_i\|$ is given the weight $\|\mathbf{a}'_i\|$. We will abbreviate $\mathsf{Vor}'_{\mathrm{W}}(i)$ by $\mathsf{Vor}'(i)$, and $\mathsf{Vor}'_{\mathrm{W}}$ by Vor'.

(In general, weighted Voronoi diagrams are used when it is desirable to use weights to alter the influence that each site has on points in its space; for more information, see for example [Aurenhammer et al., 2013, Chapter 7].)

We would like to compare the Voronoi diagrams Vor and Vor' in order to find the desired separating hyperplane. If $\mathsf{Vor} \neq \mathsf{Vor}'$, then there exists some $\mathbf{y} \in S^{n-1}$ such that $\mathbf{y} \in \mathsf{Vor}(i)$ and $\mathbf{y} \notin \mathsf{Vor}'(i)$, but instead $\mathbf{y} \in \mathsf{Vor}'(j)$, where $i \neq j$. By the definitions of the Voronoi cells, this implies that

$$\begin{aligned} \langle \mathbf{a}_i, \mathbf{y} \rangle &\geq \langle \mathbf{a}_j, \mathbf{y} \rangle \quad (\text{from } \mathsf{Vor}(i)) \\ \langle \mathbf{a}'_j, \mathbf{y} \rangle &\geq \langle \mathbf{a}'_i, \mathbf{y} \rangle \quad (\text{from } \mathsf{Vor}'(j)). \end{aligned} \tag{4}$$

This leads to the desired separating hyperplane between $\mathbf{A}$ and $\mathbf{A}_t$.

In the remainder of this section on the homogeneous case, we will consider the two remaining issues: first, how to compare Vor and Vor', and find such a point $\mathbf{y}$ if $\mathsf{Vor} \neq \mathsf{Vor}'$; and second, what to do if $\mathsf{Vor} = \mathsf{Vor}'$.

In order to compare Vor and Vor', the general idea is to use our knowledge of Vor', along with information provided about Vor by the furthest oracle, to test whether facets of a cell $\mathsf{Vor}'(i)$ are also facets of a cell $\mathsf{Vor}(i)$, for every cell $i = 1, \ldots, m$. (A facet of an n-dimensional Voronoi cell is an $(n-1)$-dimensional bound or "border" of the Voronoi cell. For example, in Figure 2, the facets of the Voronoi cells are lines.)

For each facet of the cell $\mathsf{Vor}'(i)$, we can find points $\mathbf{y}$ and $\boldsymbol{\epsilon}_\mathbf{y}$ such that

$$\begin{aligned} \mathbf{y} &\in \mathsf{Vor}'(i) \cap \mathsf{Vor}'(j) \\ \mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y} &\in \mathsf{Vor}'(i) \setminus \mathsf{Vor}'(j) \\ \mathbf{y} - \boldsymbol{\epsilon}_\mathbf{y} &\in \mathsf{Vor}'(j) \setminus \mathsf{Vor}'(i), \end{aligned}$$

for some $j \neq i$. If we send the points $\mathbf{y} + \boldsymbol{\epsilon}_\mathbf{y}$ and $\mathbf{y} - \boldsymbol{\epsilon}_\mathbf{y}$ to the oracle and receive the answers i and j, respectively, then we conclude that $\mathbf{y}$ is in the facet of $\mathsf{Vor}(i)$ and $\mathsf{Vor}(j)$ in the Voronoi diagram Vor. If this is repeated $n-1$ times, with linearly independent vectors $\mathbf{y}_1, \ldots, \mathbf{y}_{n-1}$, and the oracle sends the expected answer each time, then we conclude that this facet between $\mathsf{Vor}'(i)$ and $\mathsf{Vor}'(j)$ in Vor' is also a facet between $\mathsf{Vor}(i)$ and $\mathsf{Vor}(j)$ in Vor. If at some point, the oracle returns an unexpected answer, the points $\mathbf{y}$ and $\boldsymbol{\epsilon}_\mathbf{y}$ can be used to construct a separating hyperplane between $\mathbf{A}$ and $\mathbf{A}_t$ as in (4).

If the procedure described above checks all facets of all cells of Vor' and always receives the expected answers from the oracle, then we conclude that $\mathsf{Vor}' = \mathsf{Vor}$. In this case, we cannot construct a separating hyperplane, but we can use the fact that the procedure has obtained Vor to recover the rows of $\mathbf{A}$. First, we use the fact that, given a Voronoi diagram Vor, a set of points generating Vor can be computed efficiently [Hartvigsen, 1992]. Secondly, we use that the set of points generating Vor is in fact unique [Ash and Bolker, 1985]. Therefore, given Vor, it is possible to recover $\mathbf{A}$, and find a point satisfying the homogeneous constraints $\mathbf{Ax} \geq \mathbf{0}$.

G.2.3 Inhomogeneous Case

In the inhomogeneous case, we consider a linear program with constraints $\mathbf{Ax} \geq \mathbf{b}$. Once again, the intention is to use the ellipsoid method to search for $(\mathbf{A}, \mathbf{b})$, such that the center of the ellipsoid at time t is $(\mathbf{A}_t, \mathbf{b}_t)$. If the constraints $\mathbf{A}_t\mathbf{x} \geq \mathbf{b}_t$ are feasible, then a point $\mathbf{x}$ satisfying the constraints is sent to the oracle as a possible feasible point for the linear program with constraints $\mathbf{Ax} \geq \mathbf{b}$, and the index received from the oracle is used to construct a separating hyperplane. Therefore, in this section, we assume that there is no point $\mathbf{x}$ satisfying $\mathbf{A}_t\mathbf{x} \geq \mathbf{b}_t$, but it is still necessary to find a separating hyperplane between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}_t, \mathbf{b}_t)$ in order to continue the ellipsoid method. We again assume without loss of generality that $\|\mathbf{a}_1\| = \cdots = \|\mathbf{a}_m\| = 1$.

Returning to (3), we see that if the proposed point $\mathbf{x}$ does not satisfy $\mathbf{Ax} \geq \mathbf{b}$, then the point $\mathbf{x}$ is such that

$$\mathbf{x} \in \{\mathbf{p} \in \mathbb{R}^n : b_i - \langle \mathbf{a}_i, \mathbf{p} \rangle \geq b_j - \langle \mathbf{a}_j, \mathbf{p} \rangle, j = 1, \ldots, m\}. \tag{5}$$

The set on the right-hand side of (5) will be interpreted as part of a *generalized*, *farthest* Voronoi diagram, a concept which we now briefly introduce with definitions in the simple case of the plane, similar to Definition 3.

First, a *farthest* Voronoi diagram is similar to a regular (or *closest*) Voronoi diagram, except points belong to a given farthest Voronoi cell when they are farther away from the relevant point than they are from any other point. We state this more formally in the following definition.

Definition 4. *Let $P \coloneqq \{p_1, p_2, \ldots, p_n\} \subset \mathbb{R}^2$ be a set of distinct points in the plane, equipped with the Euclidean metric d. Let $I_n = \{1, 2, \ldots, n\}$. We define the farthest Voronoi polygon, or farthest Voronoi cell, associated with p_i to be the set*

$$\mathsf{Vor}(i) \coloneqq \left\{p \in \mathbb{R}^2 \colon d(p, p_i) \geq d(p, p_j) \text{ for } j \in I_n \setminus \{i\}\right\},$$

and the set

$$\mathsf{Vor}(P) \coloneqq \{\mathsf{Vor}(1), \mathsf{Vor}(2), \ldots, \mathsf{Vor}(n)\}$$

to be the farthest Voronoi diagram generated by P.

Next, we note that there is in general no requirement that the points (or "sites") generating a Voronoi diagram be singletons. They may in general be subsets of the space.

Definition 5. *Let $P \coloneqq \{P_1, P_2, \ldots, P_n\}$ be a set of subsets of $\mathbb{R}^2$, so $P_i \subset \mathbb{R}^2$ for $i = 1, \ldots, n$. We define the distance between point $x \in \mathbb{R}^2$ and set $A \subset \mathbb{R}^2$ to be*

$$d(x, A) \coloneqq \inf \{d(x, a) \colon a \in A\}.$$

We define the generalized Voronoi cell associated with P_i to be the set

$$\mathsf{GenVor}(i) \coloneqq \left\{p \in \mathbb{R}^2 \colon d(p, P_i) \leq d(p, P_j) \text{ for } j \in I_n \setminus \{i\}\right\}.$$

We define the set

$$\mathsf{GenVor} \coloneqq \{\mathsf{GenVor}(1), \mathsf{GenVor}(2), \ldots, \mathsf{GenVor}(n)\}$$

to be the generalized Voronoi diagram generated by P.

Both definitions given above, like Definition 3, can be generalized to any metric space (X, d). For more information on farthest Voronoi diagrams and generalized Voronoi diagrams, see for example [Okabe et al., 2000, Aurenhammer et al., 2013].

In light of Definitions 4 and 5, we see that

$$\{\mathbf{p} \in \mathbb{R}^n \colon b_i - \langle \mathbf{a}_i, \mathbf{p} \rangle \geq b_j - \langle \mathbf{a}_j, \mathbf{p} \rangle, j = 1, \ldots, m\}$$

is the set of points $\mathbf{p} \in \mathbb{R}^n$ such that $\mathbf{p}$ is farther from the half-space $\{\mathbf{z} \in \mathbb{R}^n \colon \langle \mathbf{a}_i, \mathbf{z} \rangle \geq b_i\}$ than from any other half-space $\{\mathbf{z} \in \mathbb{R}^n \colon \langle \mathbf{a}_j, \mathbf{z} \rangle \geq b_j\}$. Thus, we see that if a proposed point $\mathbf{x}$ results in the feedback i from the oracle, then

$$\mathbf{x} \in \{\mathbf{p} \in \mathbb{R}^n \colon b_i - \langle \mathbf{a}_i, \mathbf{p} \rangle \geq b_j - \langle \mathbf{a}_j, \mathbf{p} \rangle, j = 1, \ldots, m\} =: \mathsf{GenVor}(i),$$

and there is an associated generalized, farthest Voronoi diagram

$$\mathsf{GenVor} \coloneqq \{\mathsf{GenVor}(1), \mathsf{GenVor}(2), \ldots, \mathsf{GenVor}(m)\}.$$

The next part of the procedure is similar to the homogeneous case. Given $(\mathbf{A}_t, \mathbf{b}_t)$, there is an associated generalized, farthest Voronoi diagram GenVor', with cells

$$\mathsf{GenVor}'(i) \coloneqq \left\{\mathbf{p} \in \mathbb{R}^n \colon b'_i - \langle \mathbf{a}'_i, \mathbf{p} \rangle \geq b'_j - \langle \mathbf{a}'_j, \mathbf{p} \rangle, j = 1, \ldots, m\right\},$$

and we would like to compare GenVor and GenVor'. As in the homogeneous case, if $\mathsf{GenVor} \neq \mathsf{GenVor}'$, then there exists some $\mathbf{y} \in \mathbb{R}^n$ such that $\mathbf{y} \in \mathsf{GenVor}(i)$ and $\mathbf{y} \in \mathsf{GenVor}'(j)$ where $i \neq j$. Therefore, by the definitions of the generalized Voronoi cells,

$$\begin{aligned} b_i - \langle \mathbf{a}_i, \mathbf{y} \rangle &\geq b_j - \langle \mathbf{a}_j, \mathbf{y} \rangle \quad (\text{from } \mathsf{GenVor}(i)) \\ b'_j - \langle \mathbf{a}'_j, \mathbf{y} \rangle &\geq b'_i - \langle \mathbf{a}'_i, \mathbf{y} \rangle \quad (\text{from } \mathsf{GenVor}'(j)) \end{aligned}$$

and these inequalities lead to the desired separating hyperplane between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}_t, \mathbf{b}_t)$.

Using a similar procedure to that employed in the homogeneous case, we can check each facet of each cell of GenVor', and either find a point $\mathbf{y}$ leading to a separating hyperplane, or we conclude that $\mathsf{GenVor}' = \mathsf{GenVor}$.

A significant complication in the inhomogeneous case is that concluding that $\mathsf{GenVor}' = \mathsf{GenVor}$ does not imply that $\mathbf{A}$ and $\mathbf{b}$ can be recovered. To work around this issue and find the desired separating hyperplane between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}_t, \mathbf{b}_t)$, we need to introduce the following concepts.

Definition 6. *Given $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^m$, we define*

$$d(\mathbf{A}, \mathbf{b}) \coloneqq \min_{\mathbf{x} \in \mathbb{R}^n} \left\{ \max_{i=1,\ldots,m} \{b_i - \langle \mathbf{a}_i, \mathbf{x} \rangle\} \right\},$$

$$\mathsf{Extreme}(\mathbf{A}, \mathbf{b}) \coloneqq \left\{ \mathbf{x} \in \mathbb{R}^n \colon \max_{i=1,\ldots,m} \{b_i - \langle \mathbf{a}_i, \mathbf{x} \rangle\} = d(\mathbf{A}, \mathbf{b}) \right\}.$$

Informally, $d(\mathbf{A}, \mathbf{b})$ is the "smallest largest distance" that is possible to the half-spaces defined by $\mathbf{A}\mathbf{x} \geq \mathbf{b}$. The set of extreme points $\mathsf{Extreme}(\mathbf{A}, \mathbf{b})$ is informally the set of vectors having such a smallest largest distance.

Definition 7. *Given $\mathbf{A} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$ and $\mathbf{p} \in \mathbb{R}^n$, we define the support linear system of $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ at $\mathbf{p}$ to be*

$$\{\langle \mathbf{a}_i, \mathbf{x} \rangle \geq 0 \colon i \in S\},$$

where S is the set of indices of the half-spaces of $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ that have the same maximal distance to $\mathbf{p}$.

It is important to note for what follows that the support linear system is homogeneous, because this will allow us to apply techniques from the previous section.

Lemma 8 from [Bei et al., 2013] provides the framework for the use of the concepts introduced above to find the desired separating hyperplane between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}_t, \mathbf{b}_t)$, or to determine that the constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ cannot be satisfied. The lemma states that the constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ are infeasible if and only if there is a point $\mathbf{p} \in \mathbb{R}^n$ such that the following two conditions are true:

1. $\mathbf{p}$ does not satisfy $\mathbf{A}\mathbf{x} \geq \mathbf{b}$
2. The support linear system of $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ at $\mathbf{p}$ is infeasible.

If $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ is infeasible, then furthermore, any point in $\mathsf{Extreme}(\mathbf{A}, \mathbf{b})$ satisfies the above two conditions.

We now describe how Lemma 8 is applied. First, we find a point $\mathbf{p} \in \mathsf{Extreme}(\mathbf{A}_t, \mathbf{b}_t)$. We propose $\mathbf{p}$ to the oracle. If the oracle returns Yes, the procedure terminates. If the oracle returns index i, then following Lemma 8, we investigate the support linear system of $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ at $\mathbf{p}$. We will propose points $\mathbf{x}$ to the oracle that are very near $\mathbf{p}$. In particular, if $\mathbf{x}' \in \mathbb{R}^n$ is such that $\|\mathbf{x}'\| \leq \epsilon$, then $x = \mathbf{p} + \mathbf{x}' \in \{\mathbf{z} \in \mathbb{R}^n \colon \|\mathbf{z} - \mathbf{p}\| \leq \epsilon\} =: \mathbb{B}^{\epsilon}_{\mathbf{p}}$. When such a point $\mathbf{x}$ is proposed to the oracle, and the feedback i is received, we observe that

$$\begin{aligned}
i &\in \operatorname*{arg\,max}_{j \in \{1,\ldots,m\}} \{b_j - \langle \mathbf{a}_j, \mathbf{x} \rangle\} \\
&= \operatorname*{arg\,max}_{j \in S} \{b_j - \langle \mathbf{a}_j, \mathbf{x} \rangle\} \\
&= \operatorname*{arg\,max}_{j \in S} \{b_j - \langle \mathbf{a}_j, \mathbf{p} \rangle - \langle \mathbf{a}_j, \mathbf{x}' \rangle\} \\
&= \operatorname*{arg\,max}_{j \in S} \{-\langle \mathbf{a}_j, \mathbf{x} \rangle\}.
\end{aligned}$$

The set membership is by definition of the furthest oracle. The first equality occurs because ϵ is sufficiently small. The second equality occurs because $\mathbf{x} = \mathbf{p} + \mathbf{x}'$ and using bilinearity of the inner product. The final equality occurs because all half-spaces with indices in S have the same maximum distance to $\mathbf{p}$.

Based on the above reasoning, we conclude that within $\mathbb{B}^{\epsilon}_{\mathbf{p}}$, GenVor is equal to the weighted spherical Voronoi diagram corresponding to the support system. By the previous section, we are then able to identify the rows $\mathbf{a}_j$ corresponding to the indices $j \in S$.

Having identified the support system, we check if it is feasible. If it is feasible, there exists a point $\mathbf{x}^*$ such that $\langle \mathbf{a}'_i, \mathbf{x}^* \rangle \leq 0 < \langle \mathbf{a}_i, \mathbf{x}^* \rangle$, which gives a separating hyperplane between $(\mathbf{A}, \mathbf{b})$ and $(\mathbf{A}_t, \mathbf{b}_t)$. On the other hand, if the support system is not feasible, then by Lemma 8, the constraints $\mathbf{A}\mathbf{x} \geq \mathbf{b}$ cannot be satisfied, and the procedure terminates.

H Empirical Example

Consider the following CMDP: $\mathcal{S} = 2$, $\mathcal{A} = 2$, $K = 1$, $\gamma = 0$,

$$P = \left[\begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \quad \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} \right], \quad r = \begin{bmatrix} 10.0 & 10.0 \\ 10.0 & 10.0 \end{bmatrix}, \quad c = \begin{bmatrix} 5.0 & 1.0 \\ 1.0 & 1.0 \end{bmatrix}$$

Per the linear program formulation in Section 4, we construct A and b as follows:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 5 & 1 & 1 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

We approximate A and b as $A^{'}$ and $b^{'}$. In this example assume that both are correct approximations except for $b^{'}$ which incorrectly approximates C_1 as 4 rather than 2 in the $^{\text{th}}$ index.

$$A' = \begin{bmatrix} 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ 5 & 1 & 1 & 1 \end{bmatrix}, \quad b' = \begin{bmatrix} 0.5 \\ -0.5 \\ 0.5 \\ -0.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}$$

We find a feasible solution for the equation $A^{'}x \leq b^{'}$ where $x = [0.5, 0, 0, 0]$. Upon proposing this solution to the oracle, we receive the index of the violated constraint: 8. Per the ellipsoid method, we halve the $^{\text{th}}$ index of $b^{'}$ and find a feasible solution: $x = [0.375, 0.125, 0, 0]$. Upon proposing this newly constructed solution to the oracle, it replies Yes, indicating that x is a feasible solution for the system $Ax \leq b$