Solving Linear Programming with Constraints Unknown

Xiaohui Bei[1], Ning Chen[2], and Shengyu Zhang[3]

[1] Max Planck Institute for Informatics, Germany
[2] Nanyang Technological University, Singapore.
[3] The Chinese University of Hong Kong, Hong Kong.

Abstract. What is the value of input information in solving linear programming? The celebrated ellipsoid algorithm tells us that the full information of input constraints is not necessary; the algorithm works as long as there exists an oracle that, on a proposed candidate solution, returns a violation in the form of a separating hyperplane. Can linear programming still be efficiently solved if the returned violation is in other formats?

Motivated by some real-world scenarios, we study this question in a trial-and-error framework: there is an oracle that, upon a proposed solution, returns the *index* of a violated constraint (with the content of the constraint still hidden). When more than one constraint is violated, two variants in the model are investigated. (1) The oracle returns the index of a "most violated" constraint, measured by the Euclidean distance of the proposed solution and the half-spaces defined by the constraints. In this case, the LP can be efficiently solved (under a mild condition of non-degeneracy). (2) The oracle returns the index of an arbitrary (i.e., worst-case) violated constraint. In this case, we give an algorithm with running time exponential in the number of variables. We then show that the exponential dependence on n is unfortunately necessary even for the query complexity. These results put together shed light on the amount of information that one needs in order to solve a linear program efficiently.

The proofs of the results employ a variety of geometric techniques, including the weighted spherical Voronoi diagram and the furthest Voronoi diagram.

1 Introduction

Solving linear programming (LP) is a central question studied in operations research and theoretical computer science. The existence of efficient algorithms for LP is one of the cornerstones of a broad class of designs in, for instance, approximation algorithms and combinatorial optimization. The feasibility problem of linear programming asks to find an $x \in \mathbb{R}^n$ to satisfy a number of linear constraints $Ax > b$. Some previous algorithms, such as the simplex and interior point algorithms, assume that the constraints are explicitly given. In contrast, the ellipsoid method is able to find a feasible solution even without full knowledge of the constraints. This remarkable property grants the ellipsoid method an important role in many theoretical applications.

A central ingredient in the ellipsoid method is an oracle that, for a proposed (infeasible) point $x \in \mathbb{R}^n$, provides a violation that separates x and the feasible region of the LP in the format of a hyperplane. Such a separation oracle captures situations in which the input constraints are unavailable or cannot be accessed affordably, and the available information is from separating hyperplanes for proposed solutions. A natural question is

what if the feedback for a proposed solution is not a separating hyperplane. Aside from theoretical curiosity, the question relates to practical applications, where the acquired violation information is actually rather different and even more restricted and limited.

Transmit power control in cellular networks has been extensively studied in the past two decades, and the techniques developed have become foundations in the CDMA standards in today's 3G networks. In a typical scenario, there are a number of pairs of transmitters and receivers, and the transmission power of each transmitter needs to be determined to ensure that the signal is strong enough for the target receiver, yet not so strong that it interferes with other receivers. This requirement can be written as an LP of the form $Ax > b$, where each constraint i corresponds to the requirement that the Signal to Interference Ratio (SIR) is no less than a certain threshold. In general the power control is a well-known hard problem (except for very few cases, such as power minimization [9]); a major difficulty is that matrix A depends mainly on the "channel gains", which are largely unknown in many practical scenarios [4]. Thus the LP $Ax > b$ needs to be solved despite the unavailability of (A, b). What is available here is that the system can try some candidate solution x and observe violation information (namely whether the SIR exceeds the threshold). The system can then adjust and propose new solutions until finally finding an x to satisfy $Ax > b$.

There are more examples in other areas (e.g., normal form games and product design and experiments [20]) with input information hidden. In these examples, for any proposed solution that does not satisfy all the constraints, only certain salient phenomena of violation (such as signal interference) are exhibited, which give *indices* of violated constraints only. With so little information obtained from violations, is it still possible to solve LP efficiently? In general, what is the least amount of input information, in what format, that one needs to solve a linear program efficiently? This work attempts to address these questions on the value of input information in solving LP.

1.1 Model and Results

Our model is defined as follows. In an LP $Ax > b$, the constraints $a_i x > b_i$ are hidden from us. We can propose candidate solutions $x \in \mathbb{R}^n$ to a *verification oracle*[4]. If x satisfies $Ax > b$, then the oracle returns Yes and the job is done. If x is not a feasible solution, then the oracle returns the index of a violated constraint. The algorithm continues until it either finds a feasible solution or concludes that no feasible solution exists. The algorithm is adaptive in the sense that future queries may depend on the information returned during previous queries. We focus only on the feasibility problem, to which an optimization LP can be transformed by a standard binary search.

Note that when the proposed solution is not feasible, the oracle returns only the *index* i of a violation rather than the constraint $a_i x > b_i$ itself. We make this assumption for two reasons. First, consistent with the aforementioned examples, we are often only able to observe unsatisfactory phenomena (such as a strong interference in the power control problem). However, the exact reasons (corresponding to the content of violated constraints) for these problems may still be unknown. Second, as our major focus is on the value of information in solving linear programming, a weaker assumption on the information obtained implies stronger algorithmic complexity results. Indeed, as will

[4] The verification oracle is simply a means of determining whether a solution is feasible. It arises from the nature of LP as shown from the foregoing examples. For infeasible solutions, the feedback is a signaled violation.

be shown, in some settings efficient algorithms exist even with this seeming deficit of information.

For a proposed solution x, if there are multiple violated constraints, the oracle returns the index of one of them[5]. This raises the question of which violation the oracle returns, and two variants are studied in this paper. In the first one, the oracle gives more information by returning the index of a "most violated" constraint, where the extent of a violation is measured by $(b_i - \langle a_i, x\rangle)/\|a_i\|$, the Euclidean distance of the proposed solution x and the half-space defined by the constraint. This oracle, referred to as the *furthest oracle*, attempts to capture the situation in which the first violation that occurs or is observed is usually the most severe and dominant one. The second variant follows the tradition of worst-case analysis in theoretical computer science, and makes no assumption about the returned violation. This oracle is referred to as the *worst-case oracle*.

We will denote by UnknownLP the problem of solving LP with unknown constraints in the above model. In either oracle model, the time complexity is the minimum amount of time needed for any algorithm to solve the UnknownLP problem, where each query, as in the standard query complexity, costs a unit of time.

Our results are summarized below. In a nutshell, when given a furthest oracle, a polynomial-time algorithm exists to solve LP (under a mild condition of non-degeneracy). On the other hand, if only a worst-case oracle is given, the best time cost is polynomial in m, the number of constraints, but exponential in n, the number of variables. Note that it is efficient when n is small, a well-studied scenario called *fixed-dimensional LP*. The exponential dependence on n is unfortunately necessary even for the query complexity. This lower bound, when combined with the positive result for the furthest oracle case, yields an illustration of the boundary of tractable LP.

Theorem 1 *The* UnknownLP *problem can be solved in time polynomial in the input size*[6] *in the furthest oracle model, provided that the input is non-degenerate.*

The exact definition of non-degeneracy is given in Section 3. The condition is mild; actually a random perturbation on inputs yields non-degeneracy, thus the theorem implies that the smooth complexity is polynomial.

The main idea of the algorithm design is as follows. Instead of searching for a solution x directly, we consider the point $(A, b) \in \mathbb{R}^{m(n+1)}$ as a degenerate polyhedron, and use the ellipsoid method to find (A, b). In each iteration take the center (A', b') of the current ellipsoid in $\mathbb{R}^{m(n+1)}$, and aim to construct a separating hyperplane between (A, b) and (A', b') through queries to the furthest oracle. The main difficulty lies in the case when (A', b') is infeasible, in which a separating hyperplane cannot be constructed explicitly. It can be observed that upon a query x, with the help of the furthest oracle, the information returned from the oracle has a strong connection to the Voronoi diagram. Specifically, if x is not a feasible solution, then the returned index is always the furthest Voronoi cell that contains x. We can manage to compute the Voronoi diagram, but this does not uniquely determine the constraints that define the LP. To handle this difficulty, we

[5] It is also natural to consider the case where the oracle returns the indices of all violated constraints. That model turns out to be so strong as to make the linear program easily solvable. By moving the proposed points and observing the change of the set of violated constraints, one can quickly identify the value of each (a_i, b_i).

[6] The notion of input size in the unknown input setting is explained in Section 2.

give a sufficient and necessary characterization reducing the input LP to that of a new and homogeneous LP, for which the constraints can be identified using the structure of a corresponding weighted spherical closest Voronoi diagram.

For the worst-case oracle, we first establish the following upper bound which is exponential in the number of variables only.

Theorem 2 *The* UnknownLP *problem with m constraints, n variables, and input size L can be deterministically solved in time $(mnL)^{poly(n)}$. In particular, the algorithm is of polynomial time for constant dimensional LP (i.e. constant number of variables n).*

At the heart of the efficiency guarantee of our algorithm is a technical bound of $\sum_{i=0}^{n}\binom{m}{i}$ on the number of "holes" formed by the union of m convex bodies in $\mathbb{R}^n$.

The above theorem implies a polynomial time algorithm when the number n of variables is a constant. This is a well-studied scenario, called *fixed dimensional LP* in which n is much smaller than the number of constraints m; see [15, 5, 18, 19, 8] and the survey [7].

On the other hand, a natural question is whether the exponential dependence is necessary; at the very least, can we improve the bound to subexponential, as Kalai [15] and Matoušek et al. [18] have done for simplex-like algorithms? Unfortunately, the next lower bound theorem indicates that this is impossible.

Theorem 3 *Any algorithm that solves the* UnknownLP *problem with m constraints and n variables needs $\Omega\big(m^{\lfloor n/2\rfloor}\big)$ queries to the oracle, regardless of its time cost.*

We prove this by constructing a family of $2k = \Theta(m^{\lfloor n/2\rfloor})$ LPs, such that the first k LPs P_i have disjoint feasible regions, and the last k LPs P_i' are the infeasible variants of the first k LPs. Unless the algorithm proposes a point in one of feasible regions, the oracle is designed to return a fixed (thus meaningless) constraint index. After $k-1$ queries, the algorithm still cannot distinguish LP P_i and P_i' for some i. Thus even the feasibility problem cannot be solved with less than k queries.

It is worth comparing the exponential hardness of UnknownLP with the complexities of Nash and CE, the problems of finding a Nash or correlated equilibrium in a normal-form game, in the trial-and-error model. In our previous work [2], we presented algorithms with *polynomial* numbers of queries for Nash and CE with unknown payoff matrices in the model with worst-case oracle[7]. Nash and CE can be written as quadratic and linear programs, respectively, but why is the general UnknownLP hard while the unknown-input Nash and CE are easy (especially when all are given unlimited computational power)? The most critical reason is that in normal-form games, there *always exists* a Nash and a correlated equilibrium, but a general linear program may not have feasible solutions. Indeed, if a feasible solution is guaranteed to exist (even for only a random instance), such as when the number of constraints is no more than that of variables, then an efficient algorithm for UnknownLP does exist: see the full version [3]. (In our algorithms for UnknownLP, the major effort is devoted to handling infeasible LP instances.) It is interesting to see that the *solution-existing* property plays a fundamental role in developing efficient algorithms.

Related work. There were a few work studying LP with restricted input information [22, 23, 25], in settings different than the current paper; see the full version [3] for detailed

[7] An algorithm proposes a candidate equilibrium and a verification oracle returns the index of an arbitrary better response of some player as a violation.

comparisons. The trial-and-error model was proposed in [2], where a number of specific questions were studied. In [14], the model of [2] is extended to probabilistic queries and systematic studies about Constraint Satisfaction Problems (CSP) are conducted.

2 Preliminaries

Consider the following linear program (LP): $Ax > b$, where $A = (a_{ij})_{m\times n} \in \mathbb{R}^{m\times n}$ and $b = (b_1, \ldots, b_m)^T \in \mathbb{R}^m$. The *feasibility* problem asks to find a feasible solution $x \in \mathbb{R}^n$ that satisfies $Ax > b$ (or report that such a solution does not exist). Equivalently, this is to find a point $x \in \mathbb{R}^n$ that satisfies m linear constraints $\{a_i x > b_i : i \in [m]\}$, where each $a_i = (a_{i1}, \ldots, a_{in})$.

In the *unknown-constraint LP feasibility* problem, denoted by UnknownLP, the coefficient matrix A and the vector b are unknown to us, and we need to determine whether the LP has a feasible solution and find one if it does. We can propose candidate solutions $x \in \mathbb{R}^n$ to a *verification oracle*. If a query x is indeed a feasible solution, the oracle returns Yes and the problem is solved. Otherwise, the oracle returns an index i satisfying $a_i x \leq b_i$, i.e., the index of a violated constraint. Note that from this, the algorithm knows only the *index* i, but not a_i and b_i. In addition, if multiple constraints are violated, only the index of *one* of them is returned.

We will analyze the complexity for two types of oracles: the *furthest oracle* which returns the index of a "most" violated constraint (Section 3), and the *worst-case oracle* which can return an arbitrary index among those violated constraints (Section 4). In either variant, a query to the oracle takes unit time.

Input size and solution precision. A clarification is needed for the size of the input. Since the input LP instance (A, b) is unknown, neither do we know its binary size. To handle this issue, we assume that we are given the information that there are m constraints[8], n variables, and the binary size of the input instance (A, b) is at most L. Note that L is $O(mn\log(N))$, where N is the maximum entry (in absolute value) in A and b. We say that an algorithm solves UnknownLP efficiently if its running time is $poly(m, n, L)$.

Given an LP with input size $L = O(mn\log(N))$, it is known [16] that if the LP has a feasible solution, then there is one whose numerators and denominators of all components are bounded by $(nN)^n$. Hence, an alternative way to describe our assumption is that, instead of knowing the input size bound L, there is a required precision for feasible solutions. That is, we only look for a feasible solution in which the numerators and denominators of all components are bounded by the required precision. These two assumptions, i.e., giving an input size bound and giving a solution precision requirement, are equivalent, and it is necessary to have one of them in our algorithms.[9] In the rest of the paper, we will use the first one, the input size bound, to analyze the running time of our algorithms.

[8] Indeed, the number of constraints can be unknown to us as well: In an algorithm, we only need to track those violated constraints that have ever been returned by the oracle.

[9] Otherwise, we may not be able to distinguish between cases when there are no feasible solutions (e.g., $x > 0, x < 0$) and when there are feasible solutions but the feasible set is very small (e.g., $x > 0, x < \epsilon$). For any queried solution $y > 0$, the oracle always returns that the second constraint is violated. However, we cannot distinguish whether it is $x < 0$ in the first LP or $x < \epsilon$ in the second LP, as ϵ can be arbitrarily small and we have no information on how small it is.

The unit sphere in $\mathbb{R}^n$ is denoted by $S^{n-1} = \{x \in \mathbb{R}^n : \|x\| = 1\}$, where, throughout this paper, $\|\cdot\|$ refers to the ℓ_2-norm. A set $C \subseteq \mathbb{R}^n$ is a *convex cone* if for any $x, y \in C$ and any $\alpha, \beta > 0$, $\alpha x + \beta y$ is also in C. The *normalized volume* of a convex cone C is defined as the ratio $v(C) = \frac{\mathsf{vol}_n(C \cap B^n)}{\frac{1}{2} \cdot \mathsf{vol}_n(B^n)}$ where B^n is the closed unit ball in $\mathbb{R}^n$ and vol_n refers to the n-dimensional volume. For any set $C \in \mathbb{R}^n$, its *polar cone* C^* is the set $C^* = \left\{y \in \mathbb{R}^n : \langle x, y\rangle \leq 0, \forall x \in C\right\}$.

Lemma 4 ([24]) *Let $C_1, C_2, \ldots, C_k$ be k closed convex cones, then $(\bigcap_i C_i)^* = conv(\bigcup_i C_i^*)$.*

It was shown in [21] (Lemma 8.14) that if an LP has a feasible solution, then the set of solutions within the ball $\left\{x \in \mathbb{R}^n : \|x\| \leq n2^L\right\}$ has volume at least $2^{-(n+2)L}$. Given this lemma, we can easily derive the following claim.

Lemma 5 *If a linear program $Ax > 0$ has a feasible solution, then the feasible region is a convex cone in $\mathbb{R}^n$ and has normalized volume no less than $2^{-(2n+3)L}$.*

3 Furthest Oracle

In this section, we will consider the UnknownLP problem $Ax > b$ with the *furthest oracle*, formally defined as follows. For a proposed candidate solution x, if x is not a feasible solution, instead of returning the index of an arbitrary (worse case) violated constraint, the oracle returns the index of a "most violated" constraint, measured by the Euclidean distance from the proposed solution x and the half-space defined by the constraint. More precisely, the oracle returns the index of a constraint which, among all i with $\langle a_i, x\rangle \leq b_i$, maximizes $\frac{b_i - \langle a_i, x\rangle}{\|a_i\|}$, the distance from x to the half-space $\{z \in \mathbb{R}^n : \langle a_i, z\rangle \geq b_i\}$. If there are more than one maximizer, the oracle returns an arbitrary one.

Compared to the worse-case oracle, the furthest oracle reveals more information about the unknown LP system, and indeed, it can help us to derive a more efficient algorithm. Our main theorem in this section is the following.

Theorem 6 *The* UnknownLP *problem $Ax > b$ with a non-degenerate matrix A in the furthest oracle model can be solved in time polynomial in the input size.*

We call a matrix $A = (a_1, \ldots, a_m)^T$ *non-degenerate* if for each point $p \in S^{n-1}$, at most n points in $\left\{\frac{a_1}{\|a_1\|}, \ldots, \frac{a_m}{\|a_m\|}\right\}$ have the same spherical distance to p on S^{n-1}. This assumption is with little loss of generality; it holds for almost all real instances and can be derived easily by a small perturbation.

Next we describe our algorithm for the special case of $Ax > 0$.

3.1 Algorithm Solving $Ax > 0$

We assume without loss of generality that $\|a_i\| = 1$ for all i. Furthermore, we can also always propose points in S^{n-1} for the same reason.

Ellipsoid method and issues. The main approach of the algorithm is to use the ellipsoid method to find the unknown matrix $A = (a_{ij})_{m \times n}$, which can be viewed as a point in the dimension $\mathbb{R}^{mn}$, i.e., a degenerate polyhedron in $\mathbb{R}^{mn}$. Initially, for the given input size information m, n and L, we choose a sufficiently large ellipsoid that contains

the candidate region of A, and pick the center $A' \in \mathbb{R}^{mn}$ of the ellipsoid. To further the ellipsoid method, we need a hyperplane separating A' from A.

Consider the linear system $A'x > 0$. If it has a feasible solution x, then $\{x : A'x > 0\}$ is a full-dimensional cone. We query an x in this cone to the oracle. If the oracle returns an affirmative answer, then x is a feasible solution of $Ax > 0$ as well, and the job is done. Otherwise, the oracle returns an index i, meaning that $\langle a_i, x\rangle \leq 0$. Hence, we have $\langle a'_i, x\rangle > 0 \geq \langle a_i, x\rangle$, which defines a separating hyperplane between A and A' with normal vector $(\underbrace{0, \ldots, 0}_{(i-1)n}, x, \underbrace{0, \ldots, 0}_{(m-i)n})$ (note that a hyperplane in $\mathbb{R}^{mn}$ has a normal vector of dimension mn, and x is a vector of dimension n, also that we know the information of A' and x). Thus, we can cut the candidate region of A by a constant fraction and continue with the ellipsoid method.

Note that there is a small issue: In our problem, the solution polyhedron degenerates to a point $A \in \mathbb{R}^{mn}$ and has volume 0. As the input A is unknown, we cannot use the standard approach in the ellipsoid method to introduce a positive volume for the polyhedron by adding a small perturbation. This issue can be handled by a more involved machinery developed by Grötschel, Lovász, and Schrijver [11, 12], which solves the strong nonemptiness problem for well-described polyhedra given by a strong separation oracle, as long as a *strong separation oracle* exists. In the algorithms described below, we will construct such oracles, thereby circumventing the issue of perturbation of the unknown point A. The same idea has been used in [2] to find a Nash equilibrium when the payoff matrix is unknown and degenerates to a point in a high-dimensional space. More discussions refer to [11, 12, 2].

The main difficulty is when the LP $A'x > 0$ is infeasible. In the following part of this section we will discuss how to find a proper separating hyperplane in this case.

Spherical (closest) Voronoi diagram. Note that $Ax > 0$ is equivalent to $-Ax < 0$, and i minimizes $\langle a_i, x\rangle$ if and only if it maximizes $\langle -a_i, x\rangle$. In the rest of this subsection, for notational convenience, we use $x \in S^{n-1}$ to denote a proposed solution point, and let $y = -x$. Since the distance from a proposed solution x to a half-space $\{z \in \mathbb{R}^n : \langle a_i, z\rangle \geq 0\}$ is $-\langle a_i, x\rangle = \langle a_i, y\rangle$, the oracle returns us an index $i \in \arg\max_i \{\langle a_i, y\rangle : \langle a_i, y\rangle \geq 0\}$ if x is not feasible. Note that $\|z - a_i\| \leq \|z - a_j\|$ if and only if $\langle a_i, z\rangle \geq \langle a_j, z\rangle$ for any $z \in S^{n-1}$; thus, $\langle a_i, y\rangle$ is closely related to the distance between a_i and y on S^{n-1}. That is, the oracle actually provides information about the closest Voronoi diagram of $a_1, \ldots, a_m$ on S^{n-1}.

The (closest) *Voronoi diagram* of a set of points $\{a_i\}_i$ in S^{n-1} is a partition of S^{n-1} into cells, such that each point a_i is associated with the cell $\{z \in S^{n-1} : d(z, a_i) \leq d(z, a_j), \forall j\}$, where d in our case is the spherical distance on S^{n-1}. We denote by Vor the spherical (closest) Voronoi diagram of the points $a_1, \ldots, a_m$ on S^{n-1} and denote by $\mathsf{Vor}(i)$ the cell in the diagram associated with a_i, i.e.,

$$\begin{aligned} \mathsf{Vor}(i) &= \{z \in S^{n-1} : \langle a_i, z\rangle \geq \langle a_j, z\rangle,\ \forall j \in [m]\} \\ &= \{z \in S^{n-1} : \|z - a_i\| \leq \|z - a_j\|,\ \forall j \in [m]\}. \end{aligned} \tag{1}$$

If the oracle returns i upon a query $x = -y \in S^{n-1}$, then $y \in \mathsf{Vor}(i)$.

Representation. Note that for a general (spherical) Voronoi diagram formed by m points, it is possible that some of its cells contain exponential number of vertices, which is unaffordable for our algorithm. However, in the H-representation of a convex polytope,

every cell can be represented by at most m linear inequalities, as shown in Formula (1). In the following, we will see that the information of these linear inequalities is sufficient to implement our algorithm efficiently.

Weighted spherical (closest) Voronoi diagram. For the presumed matrix A', note that it can be an arbitrary point in the space $\mathbb{R}^{mn}$ and each row in A' may not necessarily fall into S^{n-1}. Our solution is to consider a *weighted spherical Voronoi diagram*, denoted by Vor', of points $\frac{a'_1}{\|a'_1\|}, \ldots, \frac{a'_m}{\|a'_m\|}$ on S^{n-1} as follows: for each point $\frac{a'_i}{\|a'_i\|}$, its associated cell is defined as

$$\mathsf{Vor}'(i) = \big\{z \in S^{n-1} : \langle a'_i, z\rangle \geq \langle a'_j, z\rangle, \forall j \in [m]\big\}.$$

Note that Vor' is a partition of S^{n-1}; and if we assign a weight $\|a'_i\|$ to each point $\frac{a'_i}{\|a'_i\|}$, then for each point $p \in \mathsf{Vor}'(i)$, the site among $\frac{a'_1}{\|a'_1\|}, \ldots, \frac{a'_m}{\|a'_m\|}$ that has the smallest *weighted* distance to p is $\frac{a'_i}{\|a'_i\|}$.[10] Note that each cell of Vor' is defined by a set of linear inequalities (other than the unit norm requirement) and each of them can be computed efficiently.

Now we have two diagrams: Vor, which is unknown, and Vor', which can be represented efficiently using the H-representation. If $\mathsf{Vor} \neq \mathsf{Vor}'$, then there exists a point $y \in S^{n-1}$ such that $y \in \mathsf{Vor}(i)$ and $y \notin \mathsf{Vor}'(i)$. Suppose that $y \in \mathsf{Vor}'(j)$ for some $j \neq i$. According to the definition, we have $\langle a_i, y\rangle \geq \langle a_j, y\rangle$ and $\langle a'_i, y\rangle < \langle a'_j, y\rangle$; this gives us a separating hyperplane between A and A'. The questions are then (1) how to find such a point y when $\mathsf{Vor} \neq \mathsf{Vor}'$, and (2) what if $\mathsf{Vor} = \mathsf{Vor}'$.

Consistency check. In this part we will show how to check whether $\mathsf{Vor} = \mathsf{Vor}'$, and if not equal, how to find a y as above. Although we know neither the positions of points $a_1, \ldots, a_m$, nor the corresponding spherical Voronoi diagram Vor, we can still efficiently compare it with Vor', with the help of the oracle.

For each cell $\mathsf{Vor}'(i)$, assume that it has k facets (i.e., $(n-1)$-dimensional faces). Note that $k \leq m$ and that $\mathsf{Vor}'(i)$ is uniquely determined by these facets. Further, each facet is defined by a hyperplane $H'_{ij} = \{z \in S^{n-1} : \langle a'_i, z\rangle = \langle a'_j, z\rangle\}$ for some $j \neq i$. To decide whether $\mathsf{Vor} = \mathsf{Vor}'$, for each i and j such that $\mathsf{Vor}'(i) \cap \mathsf{Vor}'(j) \neq \emptyset$, we find a sufficiently small ϵ_y and three points y, $y + \epsilon_y$, $y - \epsilon_y$, such that

$$y \in \mathsf{Vor}'(i) \cap \mathsf{Vor}'(j) \subset H'_{ij}, \quad y + \epsilon_y \in \mathsf{Vor}'(i) \setminus \mathsf{Vor}'(j), \quad y - \epsilon_y \in \mathsf{Vor}'(j) \setminus \mathsf{Vor}'(i).$$

Notice that such y and ϵ_y exist and can be found efficiently. We now query points $y + \epsilon_y$ and $y - \epsilon_y$ to the oracle. If the oracle does return us the expected answers, i.e., i and j, respectively, then, with $\|\epsilon_y\|$ sufficiently small (up to $2^{-poly(L)}$), we can conclude that y must also be in the facet of $\mathsf{Vor}(i)$ and $\mathsf{Vor}(j)$ of the hidden diagram Vor. That is, $y \in H_{ij} = \{z \in S^{n-1} : \langle a_i, z\rangle = \langle a_j, z\rangle\}$. We implement the above procedure $n-1$ times to look for $n-1$ linearly independent points $y_1, \ldots, y_{n-1} \in \mathsf{Vor}'(i) \cap \mathsf{Vor}'(j)$. If the oracle always returns the expected answers i and j, respectively, for all $k = 1, \ldots, n-1$, then we know that $H_{ij} = H'_{ij}$.

The procedure described above can be implemented in polynomial time. Now we can use this approach to check all facets of all of the cells of Vor'. If none of them returns

[10] The reason of defining such a weighted spherical Voronoi diagram is that we want to have a separating hyperplane between A and $A' = (a'_1, \ldots, a'_m)^T$, rather than $\big(\frac{a'_1}{\|a'_1\|}, \ldots, \frac{a'_m}{\|a'_m\|}\big)^T$.

us an unexpected answer, we know that every facet of every cell $\mathsf{Vor}'(i)$ is also a facet of cell $\mathsf{Vor}(i)$, i.e., the set of linear constraints that defines $\mathsf{Vor}'(i)$ is a subset of those that define $\mathsf{Vor}(i)$. Thus, we have $\mathsf{Vor}(i) \subseteq \mathsf{Vor}'(i)$ for each i. Together with the fact that both Vor and Vor' are tessellations of S^{n-1}, we can conclude that $\mathsf{Vor} = \mathsf{Vor}'$.

Lemma 7 *For the hidden matrix $A \in \mathbb{R}^{mn}$ with spherical Voronoi diagram Vor and proposed matrix $A' \in \mathbb{R}^{mn}$ with weighted spherical Voronoi diagram Vor', we can in polynomial time*

- *either conclude that $\mathsf{Vor} = \mathsf{Vor}'$, or*
- *find a separating hyperplane between A and A'.*

A formal and detailed description of this consistency check procedure and its correctness proof can be found in the full version [3].

Voronoi diagram recognition. If the above process concludes that $\mathsf{Vor} = \mathsf{Vor}'$, we have successfully found the Voronoi diagram Vor (in its H-representation) for the hidden points $a_1, \ldots, a_m$. It was shown by Hartvigsen [13] that given a Voronoi diagram with its H-representation, a set of points that generates the diagram can be computed efficiently. Further, Ash and Bolker [1] showed that the set of points that generates a non-degenerate Voronoi diagram is unique. Therefore, by coupling these two results and the assumption that the input matrix A is non-degenerate, we are able to identify the positions of $a_1, \ldots, a_m$ given the computed Voronoi diagram Vor, and easily determine if the LP $Ax > 0$ has a feasible solution, and compute one if it exists.

The general case of $Ax > b$ New difficulties arise in the general case of $Ax > b$. A particular one is that, even for the non-degenerate input A, the Voronoi diagram may correspond to *multiple* sets of points a_i, which makes it hard to recover the a_i's. To handle this difficulty, we give a sufficient and necessary characterization reducing the input LP to that of a new and homogeneous LP, for which the constraints can be identified using the structure of a corresponding weighted spherical closest Voronoi diagram. We unfortunately have to leave this part to the full version [3] due to space limit.

4 Worst-Case Oracle

In this section, we consider the worst-case oracle. Recall that in this setting, the oracle plays as an adversary by giving the worst-case violation index to force an algorithm to use the maximum amount of time to solve the problem.

For any linear program $Ax > b$, we can introduce another variable y and transform the linear program into the following form:

$$Ax - by > 0, \quad y > 0$$

It is easy to check that $Ax > b$ is feasible if and only if the new LP is feasible, and the solutions of these two linear systems can be easily transformed to each other. Given the oracle for $Ax > b$, one can also get another oracle for the new LP easily. (On a query (x, y), if $y \leq 0$, return the index $m + 1$; otherwise, query x/y to the oracle for $Ax > b$.) This means that the $\mathsf{UnknownLP}$ problem of the homogeneous form $Ax > 0$ is no easier than the problem of the general form. In all the analysis of this section, we will therefore only consider the problem of form $Ax > 0$.

Geometric explanations. Let us consider the problem from a geometric viewpoint. Any matrix $A = (a_{ij})_{m\times n}$ can be considered as m points $a_1, a_2, \ldots, a_m$ in the n-dimensional space $\mathbb{R}^n$, where each $a_i = (a_{i1}, a_{i2}, \ldots, a_{in})$. The positions of these points are unknown to us. Finding a feasible solution $x \in \mathbb{R}^n$ that satisfies $Ax > 0$ is equivalent to finding an open half-space

$$H_x = \big\{y \in \mathbb{R}^n : \ \langle x, y\rangle \triangleq x_1y_1 + x_2y_2 + \cdots + x_ny_n > 0\big\}$$

containing all points a_i.

In an algorithm, we propose a sequence of candidate solutions. When a query $x \in \mathbb{R}^n$ violates a constraint i, we know that $\langle a_i, x\rangle \le 0$. Hence, a_i cannot be contained in the half-space H_x, and we are able to cut H_x off from the possible region of a_i. Based on this observation, we maintain a set $\mathsf{region}(i)$, the region of possible positions of point a_i consistent with the information obtained from the previous queries. Initially, no information is known about the position of any point; thus, $\mathsf{region}(i) = \mathbb{R}^n$ for all $1 \le i \le m$.

Let us have a closer look at these regions. For each i, suppose that $x_1^i, x_2^i, \ldots, x_k^i$ are the queried points we have made so far for which the oracle returns index i. Then all information we know about a_i till this point is that the possible region is $\mathsf{region}(i) = \bigcap_{j=1}^k \{y \in \mathbb{R}^n : \langle x_j^i, y\rangle \le 0\}$. Since $\mathsf{region}(i)$ is the intersection of k closed half-spaces, it is a convex set. Equivalently, this means that any feasible solution to the LP, if existing, cannot be in $\mathsf{region}(i)^*$, the polar cone of $\mathsf{region}(i)$. Since the polar cone of a half-space $\{y \in \mathbb{R}^n \mid \langle x_j^i, y\rangle \le 0\}$ is the ray along its normal vector, i.e., $\{\lambda x_j^i \mid \lambda \ge 0\}$, we have by Lemma 4 that

$$\mathsf{region}(i)^* = conv\Big(\bigcup_j \big\{y \mid \langle x_j^i, y\rangle \le 0\big\}^*\Big) = conv\Big(\big\{\lambda x_j^i \mid 1 \le j \le k, \lambda \ge 0\big\}\Big).$$

Since $\mathsf{region}(i)^*$'s are the forbidden areas for any feasible solution, we can conclude that the LP has no feasible solution if $\bigcup_i \mathsf{region}(i)^* = \mathbb{R}^n$.

Convex hull covering algorithms. Based on above observations, we now sketch a framework of *convex hull covering algorithms* that solves the UnknownLP problem. The algorithm maintains a list of m convex cones

$$\mathsf{region}(1)^*, \mathsf{region}(2)^*, \ldots, \mathsf{region}(m)^* \subseteq \mathbb{R}^n.$$

Initially, $\mathsf{region}(i)^* = \emptyset$ for all $1 \le i \le m$. On each query $x \in \mathbb{R}^n$, the oracle either returns Yes, indicating that the problem is solved, or returns us an index i, in which case we update $\mathsf{region}(i)^*$ to $conv\,(\mathsf{region}(i)^*, \{\lambda x \mid \lambda > 0\})$. The algorithm terminates when either the oracle returns Yes, or when $\mathbb{R}^n - \bigcup_i \mathsf{region}(i)^*$ does not contain a convex cone with normalized volume at least $2^{-(2n+3)L}$, which indicates that the given instance has no feasible solution. The above discussion can be formalized into the following theorem.

Theorem 8 *Any algorithm that falls into the convex hull covering algorithm framework solves the* UnknownLP *problem.*

Though the framework guarantees the correctness, it does not specify how to make queries to control complexity. Next we will show an algorithm with nearly optimal complexity. The basic idea is to use induction on dimension. That is, we pick an $(n-1)$-dimensional subspace and recursively solve the problem on the subspace. The subroutine

either finds a point x in the subspace that satisfies $Ax > 0$ (in which case the algorithm ends), or finds out that there is no feasible solution in the entire subspace. In the latter case, the whole space of candidate solutions can be divided into two open half-spaces, and we will work on each of them separately. In general, we have a collection of connected regions that can still contain a valid solution. These regions are the "holes", formally called *chambers*, separated by $\bigcup_i \mathsf{region}(i)^*$ (recall that points in $\mathsf{region}(i)^*$ cannot be a feasible solution). We can then pick a chamber with the largest volume, and cut it into two balanced halves by calling the subroutine on the hyperplane slicing the chamber.

There are several issues for the above approach. The main one is that there may be too many chambers: *a priori*, the number can grow exponentially with m. There are also other technical issues to be handled, such as how to represent chambers (which are generally concave), how to compute (even approximately) the volume of chambers, how to find a hyperplane to cut a chamber into two balanced halves, etc.

For the first and main issue, it can be shown that the number of chambers cannot be too large. In general, Kovalev [17] showed that any m convex sets in $\mathbb{R}^n$ cannot form more than $\sum_{i=1}^{n} \binom{m}{i}$ chambers. For the rest of the technical issues, we deal with them in the following way. Instead of keeping track of all actual chambers, in our algorithm, we maintain a collection of disjoint *sector cylinders*, which can be shown to be supersets of chambers. Furthermore, we keep only cylinders that contain at least one chamber, thus, the bound for the number of chambers also bounds the number of cylinders from above.

Theorem 2 can be proved based on the ideas described above. The details of the algorithm and its analysis can be found in the full version [3].

5 Concluding Remarks

We consider solving LP when the input constraints are unknown, and show that different kinds of violation information yield different computational complexities. LP is a powerful tool employed in real applications dealing with objects that are largely unknown. For example, in the node localization of sensor networks where the locations of targets are unknown [6], the computation of the locations in some settings can be formulated as a linear program with constraints that measure partial information obtained from data [10]. However, the estimation usually has various levels of error, which may lead to violations of the presumed constraints. Interesting questions that deserve further explorations are what can be theoretically analyzed there, and in general, what other natural formats of violations there are in linear programming and what complexities they impose.

References

1. Peter F. Ash and Ethan D. Bolker. Recognizing dirichlet tessellations. *Geometriae Dedicata*, 19(2):1985, 175-206.
2. Xiaohui Bei, Ning Chen, and Shengyu Zhang. On the complexity of trial and error. In *Proceedings of the 45th ACM Symposium on Theory of Computing*, pages 31–40, 2013.
3. Xiaohui Bei, Ning Chen, and Shengyu Zhang. Solving linear programming with constraints unknown. *arXiv:1304.1247*, 2013.
4. Mung Chiang, Prashanth Hande, Tian Lan, and Chee-Wei Tan. Power control in wireless cellular networks. *Foundations and Trends in Networking*, 2(4):381–533, 2007.
5. Kenneth L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *Journal of the ACM*, 42(2):488–499, 1995.

6. Lance Doherty, Kristofer Pister, and Laurent El Ghaoui. Convex position estimation in wireless sensor networks. In *Proceedings of the Twentieth IEEE Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pages 1655–1663, 2001.
7. Martin Dyer, Nimrod Megiddo, and Emo Welzl. Linear programming. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, 2004.
8. Martin E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.
9. Gerard J. Foschini and Zoran Miljanic. A simple distributed autonomous power control algorithm and its convergence. *IEEE Transactions on Vehicular Technology*, 42(3):641–646, 1993.
10. Camillo Gentile. Distributed sensor location through linear programming with triangle inequality constraints. In *Proceedings of IEEE Conference on Communications*, pages 3192–3196, 2005.
11. Martin Grötschel, László Lovász, and Alexander Schrijver. Geometric methods in combinatorial optimization. *Progress in Combinatorial Optimization*, pages 167–183, 1984.
12. Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.
13. David Hartvigsen. Recognizing voronoi diagrams with linear programming. *INFORMS Journal on Computing*, 4(4):369–374, 1992.
14. Gábor Ivanyos, Raghav Kulkarni, Youming Qiao, Miklos Santha, and Aarthi Sundaram. On the complexity of trial and error for constraint satisfaction problems. In *Proceedings of the 41st International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 663–675, 2014.
15. Gil Kalai. A subexponential randomized simplem algorithm. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 475–482, 1992.
16. L. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979.
17. M. Kovalev. A property of convex sets and its application. *Matematicheskie Zametki*, pages 89–99, 1988. English translation: Mathematical Notes, V.44, 537-543.
18. Jirí Matousek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4/5):498–516, 1996.
19. Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.
20. Douglas Montgomery. *Design and Analysis of Experiments*. Wiley, 7 edition, 2008.
21. Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.
22. Christos H. Papadimitriou and Mihalis Yannakakis. Linear programming without the matrix. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 121–129, 1993.
23. Ilya O. Ryzhov and Warren B. Powell. Information collection for linear programs with uncertain objective coefficients. *SIAM Journal on Optimization*, 22(4):1344–1368, 2012.
24. Lennart Sandgren. On convex cones. *Mathematica Scandinavica*, 2:19–28, 1954.
25. D. B. Yudin and A. S. Nemirovskii. Informational complexity and efficient methods for the solution of convex extremal problems. *Ekonomika i Matematicheskie Metody*, 12:357–369, 1976. English translation: Matekon 13 (3), 25-45, 1977.