Short simplex paths in lattice polytopes

Alberto Del Pia * Carla Michini †

December 11, 2019

Abstract

We consider the problem of optimizing a linear function over a lattice polytope $P$ contained in $[0,k]^n$ and defined via $m$ linear inequalities. We design a simplex algorithm that, given an initial vertex, reaches an optimal vertex by tracing a path along the edges of $P$ of length at most $O(n^6k \log k)$. The length of this path is independent on $m$ and is the best possible up to a polynomial function, since it is only polynomially far from the worst case diameter. The number of arithmetic operations needed to compute the next vertex in the path is polynomial in $n$, $m$ and $\log k$. If $k$ is polynomially bounded by $n$ and $m$, the algorithm runs in strongly polynomial time.

Key words: Lattice polytope, Simplex algorithm, Diameter, Strongly polynomial time

1 Introduction

Linear programming (LP) is one of the most fundamental types of optimization models. In a LP problem, we are given a polyhedron $P \subseteq \mathbb{R}^n$ and a cost vector $c \in \mathbb{Z}^n$, and we wish to solve the optimization problem

$$\max \{c^\top x \mid x \in P\}. \quad (1)$$

The polyhedron $P$ is explicitly given via a system of linear inequalities, i.e., $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$, where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$.

The simplex method is one of the main algorithms for LP, and has been selected as one of the most influential algorithms in the 20th century [9]. The simplex method moves from the current vertex to an adjacent one along an edge of the polyhedron, until an optimal vertex is reached, and the selection of the next vertex depends on a pivoting rule. The sequence of vertices generated by the simplex method is called the simplex path.

A natural lower bound on the length of a simplex path from $x^0$ to $x^*$ is given by the distance between these two vertices, which is defined as the minimum length of a path connecting $x^0$ and $x^*$ along the edges of the polyhedron $P$. The diameter of $P$ is the

---

*Department of Industrial and Systems Engineering & Wisconsin Institute for Discovery, University of Wisconsin-Madison, Madison, WI, USA. E-mail: delpia@wisc.edu.

†Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI, USA. E-mail: michini@wisc.edu.
largest distance between any two vertices of \( P \), and therefore it provides a lower bound on the length of a worst-case simplex path on \( P \).

In this paper, we consider a special class of linear programs where the feasible region is a lattice polytope, i.e., a polytope whose vertices have integer coordinates. These polytopes are particularly relevant in discrete optimization and integer programming, as they correspond to the convex hull of the feasible solutions of such optimization problems. In particular, a \([0,k]\)-polytope in \( \mathbb{R}^n \) is defined as a lattice polytope contained in the box \([0,k]^n\).

A result by Kitahara and Mizuno [11] implies that, for \([0,k]\)-polytopes, the length of the simplex path is upper bounded by a polynomial in \( k, n \) and \( m \), provided that we use either Dantzig’s or the best improvement pivoting rule. However, this upper bound can be quite far from being tight. In fact, it is known that the diameter of \([0,k]\)-polytopes is upper bounded by \( kn \) [12, 5, 7], which is, in particular, independent on the number \( m \) of constraints. We remark that \( m \) can be exponential in \( n \), even for \([0,1]\)-polytopes [3]. Thus there is a significant gap between the upper bound on the length of the simplex path given by Kitahara and Mizuno and the upper bound \( kn \) on the worst-case diameter for \([0,k]\)-polytopes. The main objective of this paper is to close this gap.

Our main contribution is the design of a pivoting rule for \([0,k]\)-polytopes in \( \mathbb{R}^n \) under which the length of the simplex path is upper bounded by a function in \( O(n^6k \log k) \), i.e., by a polynomial in \( n \) and \( k \) that is independent on \( m \). In particular, the length of our simplex path is only polynomially far from optimal, meaning that it is upper bounded by a polynomial function of the worst case diameter. In fact, it is known that: (i) for fixed \( k \), the diameter of lattice polytopes can grow linearly with \( n \); (ii) viceversa, for fixed \( n \), the diameter can grow almost linearly with \( k \). The behavior (i) can be seen already for \( k = 1 \), as the hypercube \([0,1]^n\) has diameter \( n \). For \( k = 2 \), it is known that there are \([0,2]\)-polytopes in \( \mathbb{R}^n \) of diameter \( \Omega(n) \) [5]. In general, for any fixed \( k \), there are lattice polytopes, called primitive zonotopes, that can have diameter in \( \Omega(n) \) [6]. Concerning (ii), it is known that for \( n = 2 \) there are \([0,k]\)-polytopes with diameter in \( \Omega(k^{2/3}) \) [2, 17, 1]. Moreover, for any fixed \( n \), there are primitive zonotopes with diameter in \( \Omega(k^{n(n+1)}) \) for \( k \) that goes to infinity [8].

Moreover, our pivoting rule is such that the number of operations needed to construct the next vertex in the simplex path is bounded by a polynomial in \( n, m, \) and \( \log k \). If \( k \) is bounded by a polynomial in \( n \) and \( m \), we obtain a strongly polynomial time simplex algorithm for \([0,k]\)-polytopes. This assumption is justified by the existence of \([0,k]\)-polytopes that, for fixed \( n \), have a diameter that grows almost linearly in \( k \). Consequently, in order to obtain a simplex algorithm that is strongly polynomial also for these polytopes, we need to assume that \( k \) is bounded by a polynomial in \( n \) and \( m \). We remark that in this paper we use the standard notions regarding computational complexity in Discrete Optimization, and we refer the reader to Section 2.4 in the book [15] for a thorough introduction.

2 Overview of the algorithm

Our goal is to study the length of the simplex path in the setting where the feasible region of problem (1) is a \([0,k]\)-polytope. As discussed above, our main contribution is the design
of a simplex algorithm that visits a number of vertices polynomial in \( n \) and \( k \). To the best of our knowledge, this question has not been previously addressed for lattice polytopes.

Since \( P \) is a lattice polytope in \([0,k]^n\), it is not hard to see that a basic simplex algorithm always reaches an optimal vertex of \( P \) by constructing a simplex path of length at most \( kn\|c\|_{\infty} \). Therefore, in order to reach our goal, we need to remove the dependency on \( \|c\|_{\infty} \) in the length of the simplex path.

Our first idea is to improve the dependence on \( \|c\|_{\infty} \) by recursively invoking the basic algorithm with finer and finer approximations of \( c \). We use this technique to design a multi-stage algorithm that constructs a simplex path to an optimal vertex of \( P \) of length at most \( kn(\lceil \log \|c\|_{\infty} \rceil + 1) \).

Our second idea builds on the multi-stage algorithm with the goal of obtaining a simplex path of length polynomial in \( n \) and \( k \) alone, thus completely removing the dependence on \( \|c\|_{\infty} \). In particular, we design an iterative algorithm which, at each iteration, identifies one constraint of \( Ax \leq b \) that is active at each optimal solution of (1). Such constraint is then set to equality, effectively restricting the feasible region of (1) to a lower dimensional face of \( P \).

At each iteration we compute a suitable approximation \( \tilde{c} \) of \( c \) and we maximize \( \tilde{c}^T x \) over the current face of \( P \). In order to solve this LP problem, we apply the multi-stage algorithm and we trace a path along the edges of \( P \). We also compute an optimal solution to the dual, which we exploit to identify a new constraint of \( Ax \leq b \) that is active at each optimal solution of (1). The final simplex path is then obtained by merging together the different paths constructed by the multi-stage algorithm at each iteration.

Our second idea, just discussed above, is inspired by Tardos’ strongly polynomial algorithm for combinatorial problems [16]. Tardos’ algorithm solves LP problems in standard form, i.e., problem (1) where \( P \) is a nonempty polyhedron of the form \( P = \{ x \in \mathbb{R}^n | Ax = b, x \geq 0 \} \), and it runs in polynomial time in the size of \( A \). The algorithm identifies at each iteration one nonnegativity constraint that is active at every optimal solution of the LP problem, and it sets the constraint to equality. In order to identify such constraint, at each iteration a projection of vector \( c \) is rounded and an auxiliary LP problem is solved. We remark that Tardos’ algorithm is not a simplex algorithm, as is does not trace a simplex path on \( P \).

Later Mizuno [13, 14] proposed a strongly polynomial dual simplex algorithm under the additional assumption that \( P \) is simple and that \( \Delta = 1 \). The main idea is to solve the auxiliary LP problems of Tardos’ algorithm with the dual simplex method, that runs in strongly polynomial time under the above assumptions [10]. We remark that the basic solutions generated by Mizuno’s algorithm might be not primal feasible.

### 3 The multi-stage simplex algorithm

In this section we describe our first two algorithms to solve the LP problem (1): the basic algorithm, and the multi-stage algorithm. The multi-stage algorithm builds on the basic algorithm. Furthermore, the multi-stage algorithm will be used as a subroutine in our main algorithm, which will be given in Section 4. We recall that we always assume that the polyhedron \( P \) is given via an external description, i.e., \( P = \{ x \in \mathbb{R}^n | Ax \leq b \} \).
All our algorithms are simplex algorithms, meaning that they explicitly construct a path along the edges of $P$ from a starting vertex $x^0$ to an optimal vertex. For this reason, we further assume that we are given a starting vertex $x^0$ of $P$. It will be convenient to consider the following oracle, which provides a way to construct the next vertex in the simplex path:

**Oracle.**

*Input:* A polytope $P$, a cost vector $c \in \mathbb{Z}^n$, and a vertex $x^t$ of $P$.

*Output:* Either a statement that $x^t$ is optimal (i.e., $x^t \in \text{argmax}\{c^\top x \mid x \in P\}$), or a vertex $x^{t+1}$ adjacent to $x^t$ with strictly larger cost (i.e., $c^\top x^{t+1} > c^\top x^t$).

**Observation 1.** An oracle call can be performed with a number of operations bounded by a polynomial in the size of $A$.

*Proof.* Denote by $A=\{x \leq b\}$ the subsystem of the inequalities of $Ax \leq b$ satisfied at equality by $x^t$. Note that the polyhedron $T := \{x \in \mathbb{R}^n \mid A=\{x \leq b\} \}$ is a translated cone with vertex $x^t$. Denote by $d^\top$ the sum of all the rows in $A$ and note that the vertex $x^t$ is the unique maximizer of $d^\top x$ over $T$. Let $T'$ the truncated cone $T' := \{x \in T \mid d^\top x \geq d^\top x^t - 1\}$ and note that there is a bijection between the neighbors of $x^t$ in $P$ and the vertices of $T'$ different from $x^t$. Therefore, in order to perform an oracle call, it suffices to solve the LP problem $\max\{c^\top x \mid x \in T'\}$. If the optimum is $x^t$, then the oracle returns that $x^t$ is optimal. Otherwise, if the optimum is a vertex $w$ of $T'$ different from $x^t$, then the oracle returns the corresponding neighbor of $x^t$ in $P$.

Using Tardos’ algorithm, the above LP problem can be solved in a number of operations that is polynomial in the size of the constraint matrix. It is simple to check that the size of such constraint matrix is polynomial in the size of $A$. \hfill \square

### 3.1 Basic algorithm

The simplest way to solve (1) is to recursively invoke the oracle with the given cost vector $c$, starting from the vertex $x^0$ in input. We formally describe this basic algorithm, which will also be used as a subroutine in our more complex algorithms that will be introduced later.

**Basic algorithm.**

*Input:* A $[0,k]$-polytope $P$, a cost vector $c \in \mathbb{Z}^n$, and a vertex $x^0$ of $P$.

*Output:* A vertex of $P$ maximizing $c^\top x$.

For $t = 0, 1, 2, \ldots$ Invoke the oracle with input $P$, $c$, and $x^t$. If the output of the oracle is a statement that $x^t$ is optimal, return $x^t$. Otherwise, let $x^{t+1}$ be the vertex of $P$ returned by the oracle.

The correctness of the basic algorithm is immediate. Next, we upper bound length of the simplex path generated by the basic algorithm.

**Observation 2.** The length of the simplex path generated by the basic algorithm is bounded by $c^\top x^* - c^\top x^0$, where $x^*$ is the vertex of $P$ in output. In particular, the length of the simplex path is bounded by $kn \|c\|_\infty$.
Proof. To show the first part of the statement, we only need to observe that each oracle call increases the objective value by at least one, since $c$ and the vertices of $P$ are integral.

The cost difference between $x^*$ and $x^0$ of $P$ can be bounded by

$$c^T x^* - c^T x^0 = \sum_{i=1}^{n} c_i (x_i^* - x_i^0) \leq \sum_{i=1}^{n} |c_i| |x_i^* - x_i^0| \leq kn \|c\|_\infty,$$

where the last inequality we use $|x_i^* - x_i^0| \leq k$ since $P$ is a $[0, k]$-polytope. This concludes the proof of the second part of the statement. \hfill \square

3.2 Multi-stage algorithm

The length of the simplex path generated by the basic algorithm is clearly not satisfactory. In fact, as we discussed in Section 1, our goal is to obtain a simplex path of length polynomial in $n$ and $k$, and therefore independent on $\|c\|_\infty$. In this section we improve this gap by giving a multi-stage algorithm that solves problem (1) by invoking the oracle at most $kn(\lceil \log \|c\|_\infty \rceil + 1)$ times. In particular, this algorithm yields a simplex path of the same length.

For ease of notation, we denote by $\ell := \lceil \log \|c\|_\infty \rceil$. The main idea of the multi-stage algorithm is to iteratively use the basic algorithm with the sequence of increasingly accurate integral approximations of the cost vector $c$ given by

$$c^t := \left\lceil \frac{c}{2^{\ell-t}} \right\rceil$$

for $t = 0, \ldots, \ell$.

Since $c$ is an integral vector, we have that $c^\ell = c$. Next, we describe our algorithm.

Multi-stage algorithm.

Input: A $[0, k]$-polytope $P$, a vertex $x^0$ of $P$, and a cost vector $c \in \mathbb{Z}^n$.

Output: A vertex of $P$ maximizing $c^T x$.

For $t = 0, \ldots, \ell$: Invoke the basic algorithm with input $P$, $c^t$, and $x^t$. Let $x^{t+1}$ be the vertex of $P$ returned by the basic algorithm. Return the vertex $x^{\ell+1}$.

The correctness of the multi-stage algorithm follows from the fact that the vector $x^{\ell+1}$ returned is the output of the basic algorithm with input $P$ and cost vector $c^\ell = c$.

In the remainder of the section we analyze the number of oracle calls performed by the multi-stage algorithm. We first derive some properties of the approximations $c^t$ of $c$.

Lemma 1. For each $t = 0, \ldots, \ell$, we have $\|c^t\|_\infty \leq 2^t$.

Proof. By definition of $\ell$, we have $|c_j| \leq \|c\|_\infty \leq 2^\ell$ for every $j = 1, \ldots, n$, hence $-2^\ell \leq c_j \leq 2^\ell$. For any $t \in \{0, \ldots, \ell\}$, we divide the latter chain of inequalities by $2^{\ell-t}$ and round up to obtain

$$-2^t = \left\lceil -2^t \right\rceil = \left\lceil \frac{-2^\ell}{2^{\ell-t}} \right\rceil \leq \left\lceil \frac{c_j}{2^{\ell-t}} \right\rceil \leq \left\lceil \frac{2^\ell}{2^{\ell-t}} \right\rceil = \left\lceil 2^t \right\rceil = 2^t.$$

\hfill \square
Lemma 2. For each $t = 1, \ldots, \ell$, we have $2\ell^{t-1} - c^t \in \{0, 1\}^n$.

Proof. First, we show that for every real number $r$, we have $2 \lceil r \rceil - \lfloor 2r \rfloor \in \{0, 1\}$. Note that $r$ can be written as $[r] + f$ with $f \in (-1, 0]$. We then have $2\lceil r \rceil = 2\lceil [r] + 2f \rceil = 2\lceil [r] \rceil + 2\lceil 2f \rceil$. Since $\lceil 2f \rceil \in \{-1, 0\}$, we obtain $2\lceil [r] \rceil - 2\lfloor r \rfloor \in \{-1, 0\}$, hence $2\lceil [r] \rceil - 2\lfloor r \rfloor \in \{0, 1\}$.

Now, let $j \in \{1, \ldots, n\}$, and consider the $j$th component of the vector $2\ell^{t-1} - c^t$. By definition, we have

$$2\ell^{t-1} - c^t_j = 2 \left\lceil \frac{c^t_j}{2\ell^{t-1}+1} \right\rceil - \left\lceil \frac{c^t_j}{2\ell^{t-1}} \right\rceil.$$  

The statement then follows from the first part of the proof by setting $r = c^t_j / 2\ell^{t-1} + 1$. □

We are ready to provide our bound on the length of the simplex path generated by the multi-stage algorithm. Even though the multi-stage algorithm uses the basic algorithm as a subroutine, we show that the simplex path generated by the multi-stage algorithm is much shorter than the one generated by the basic algorithm alone.

Proposition 1. The length of the simplex path generated by the multi-stage algorithm is bounded by $kn(\lceil \log \|c\|_\infty \rceil + 1)$.

Proof. Note that the multi-stage algorithm performs a total number of $\ell + 1 = \lceil \log \|c\|_\infty \rceil + 1$ iterations, and in each iteration it calls once the basic algorithm. Thus, we only need to show that, at each iteration, the simplex path generated by the basic algorithm is bounded by $kn$.

First we consider the iteration $t = 0$ of the multi-stage algorithm. In this iteration, the basic algorithm is invoked with input $P$, $c^0$, and $x^0$. Lemma 1 implies that $\|c^0\|_\infty \leq 1$, and from Observation 2 we have that the basic algorithm invokes the oracle at most $kn$ times.

Next, consider the iteration $t$ of the multi-stage algorithm for $t \in \{1, \ldots, \ell\}$. In this iteration, the basic algorithm is invoked with input $P$, $c^t$, and $x^t$, and outputs the vertex $x^{t+1}$. From Observation 2, we only need to show that $c^T x^{t+1} - c^T x^t \leq kn$.

First, we derive an upper bound on $c^T x^{t+1}$. By construction of $x^t$, the inequality $c^{-1} x \leq c^{-1} x^t$ is valid for the polytope $P$, thus

$$c^T x^{t+1} = \max\{c^T x \mid x \in P\} \leq \max\{c^T x \mid x \in [0, k]^n, \ c^{-1} x \leq c^{-1} x^t\}.$$  

The optimal value of the LP problem on the right-hand side is upper bounded by $2\ell^{t-1} x^t$. In fact, Lemma 2 implies $c^t \leq 2\ell^{t-1}$, hence for every feasible vector $x$ of the LP problem on the right-hand side, we have

$$c^T x \leq 2\ell^{t-1} x^t \leq 2\ell^{t-1} x^t.$$  

Thus we have shown $c^T x^{t+1} \leq 2\ell^{t-1} x^t$.

We can now show $c^T x^{t+1} - c^T x^t \leq kn$. We have

$$c^T x^{t+1} - c^T x^t \leq 2\ell^{t-1} x^t - c^T x^t = (2\ell^{t-1} - c^t)^T x^t \leq kn.$$  

The last inequality holds because, from Lemma 2, we know that $2\ell^{t-1} - c^t \in \{0, 1\}^n$, while the vector $x^t$ is in $[0, k]^n$. □
4 The iterative simplex algorithm

The length of the simplex path generated by the multi-stage algorithm still depends on \( \|c\|_\infty \), even if the dependence is now logarithmic instead of linear. In this section we present our iterative algorithm, which completely removes the dependence on \( \|c\|_\infty \). We remark that the multi-stage algorithm will be used as a subroutine in the iterative algorithm.

As in our previous algorithms, our input consists of a polytope \( P \), a cost vector \( c \in \mathbb{Z}^n \), and a vertex \( x^0 \) of \( P \). In particular, the polytope is nonempty. Furthermore, we can assume without loss of generality that \( P \) is full-dimensional (see Lemma 1 in [5]). Recall that, the polytope \( P \) is explicitly given via a system of linear inequalities, i.e., \( P = \{ x \in \mathbb{R}^n \mid Ax \leq b \} \), where \( A \in \mathbb{Z}^{m \times n} \), \( b \in \mathbb{Z}^m \). We further assume that each inequality in \( Ax \leq b \) is facet-defining, and that the greatest common divisor of the entries in each row of \( A \) is one. Both these assumptions are without loss of generality and it is well-known that we can reduce ourselves to this setting in polynomial time. For notational simplicity, in this section we let \( \alpha \) denote the largest absolute value of the entries of \( A \), and we define \( \mathcal{M} := \{1, 2, \ldots, m\} \).

Iterative algorithm.

Input: A full-dimensional \([0, k]\)-polytope \( P \), a cost vector \( c \in \mathbb{Z}^n \), and a vertex \( x^0 \) of \( P \).

Output: A vertex of \( P \) maximizing \( c^\top x \).

0. Let \( E := \emptyset \) and \( x^* := x^0 \).

1. Let \( \bar{c} \) be the projection of \( c \) onto the subspace \( \{ x \in \mathbb{R}^n \mid a_i^\top x = 0 \text{ for } i \in E \} \) of \( \mathbb{R}^n \).

   If \( \bar{c} = 0 \) return \( x^* \), otherwise go to 2.

2. Let \( \tilde{c} \in \mathbb{Z}^n \) be defined by \( \tilde{c}_i := \left\lfloor \frac{n^2 k \alpha}{\|\bar{c}\|_\infty} \bar{c}_i \right\rfloor \) for \( i = 1, \ldots, n \).

3. Consider the following pair of primal and dual LP problems:

\[
\begin{align*}
\max & \quad \tilde{c}^\top x \\
\text{s.t.} & \quad a_i^\top x \leq b_i \quad i \in [m] \setminus \mathcal{E} \\
\quad & \quad a_i^\top x = b_i \quad i \in \mathcal{E}.
\end{align*}
\]

(\( \tilde{P} \))

\[
\begin{align*}
\min & \quad b^\top y \\
\text{s.t.} & \quad A^\top y = \tilde{c} \\
\quad & \quad y_i \geq 0 \quad i \in [m] \setminus \mathcal{E}.
\end{align*}
\]

(\( \tilde{D} \))

Use the multi-stage algorithm to compute an optimal vertex \( \tilde{x} \) of \( \tilde{P} \) starting from \( x^* \). Compute an optimal solution \( \tilde{y} \) to the dual \( \tilde{D} \) such that (i) \( \tilde{y} \) has at most \( n \) nonzero components, and (ii) \( \tilde{y}_j = 0 \) for every \( j \in [m] \setminus \mathcal{E} \) such that \( a_j \) can be written as a linear combination of \( a_i, i \in \mathcal{E} \).

Let \( \mathcal{F} := \{ i \mid \tilde{y}_i > nk \} \), and let \( h \in \mathcal{F} \setminus \mathcal{E} \). Add the index \( h \) to the set \( \mathcal{E} \), set \( x^* := \tilde{x} \), and go back to step 1.

Note that the above algorithm is iterative in nature. In particular, an iteration of the algorithm corresponds to an execution of steps 1, 2, and 3.

Let us first show that an optimal solution \( \tilde{y} \) to the dual \( \tilde{D} \) with the properties stated in step 3 exists, and it can be computed efficiently.

7
Lemma 3. A vector \( \tilde{y} \) as in step 3 always exists and the number of operations needed to compute it is bounded by a polynomial in the size of \( A \).

Proof. First, we show how to compute a vector \( \tilde{y} \) that satisfies (i). Since \( (\tilde{P}) \) has an optimal solution, then so does \( (\tilde{D}) \) from strong duality. Let \( (\tilde{D})' \) be obtained from \( (\tilde{D}) \) by replacing each variable \( y_i, i \in \mathcal{E} \), with \( y_i^+ - y_i^- \), where \( y_i^+ \) and \( y_i^- \) are new variables which are required to be nonnegative. Clearly \( (\tilde{D}) \) and \( (\tilde{D})' \) are equivalent, so \( (\tilde{D})' \) has an optimal solution. Furthermore, since \( (\tilde{D})' \) is in standard form, it has an optimal solution \( \tilde{y}' \) that is a basic feasible solution. In particular, via Tardos’ algorithm, the vector \( \tilde{y}' \) can be computed in time bounded by a polynomial in the size of \( A \). Let \( \tilde{y} \) be obtained from \( \tilde{y}' \) by replacing each pair \( y_i^+, y_i^- \) with \( \tilde{y}_i := y_i^+ - y_i^- \). It is simple to check that \( \tilde{y} \) is an optimal solution to \( (\tilde{D}) \). Since \( \tilde{y}' \) is a basic feasible solution, it has at most \( n \) nonzero entries. By construction, so does \( \tilde{y} \).

Next, we discuss how to compute a vector \( \tilde{y} \) that satisfies (i) and (ii). Let problem \( (\tilde{P})' \) be obtained from \( (\tilde{P}) \) by dropping the inequalities \( a_j^\top x \leq b_j \), for \( j \in [m] \setminus \mathcal{E} \), such that \( a_j \) can be written as a linear combination of \( a_i, i \in \mathcal{E} \). Since problem \( (\tilde{P}) \) is feasible, then \( (\tilde{P}) \) and \( (\tilde{P})' \) have the same feasible region and are therefore equivalent. Let \( (\tilde{D})' \) be the dual of \( (\tilde{P})' \). Note that \( (\tilde{D})' \) is obtained from \( (\tilde{D}) \) by dropping the variables \( y_j \) corresponding to the inequalities of \( (\tilde{P}) \) dropped to obtain \( (\tilde{P})' \). Note that \( (\tilde{P})' \) has the same form of \( (\tilde{P}) \), thus, from the first part of the proof, we can compute a vector \( \tilde{y}' \) optimal to \( (\tilde{D})' \) with at most \( n \) nonzero components. Furthermore, \( \tilde{y}' \) can be computed in time bounded by a polynomial in the size of \( A \). Let \( \tilde{y} \) be obtained from \( \tilde{y}' \) by adding back the dropped components and setting them to zero. The vector \( \tilde{y} \) is feasible to \( (\tilde{D}) \), and, from complementary slackness with \( \tilde{x} \), it is optimal to \( (\tilde{D}) \). Furthermore, \( \tilde{y} \) satisfies (i) and (ii).

At each iteration of the iterative algorithm, let \( F \) be the face of \( P \) defined as

\[
F := \{ x \in \mathbb{R}^n \mid a_i^\top x \leq b_i \text{ for } i \in [m] \setminus \mathcal{E}, a_i^\top x = b_i \text{ for } i \in \mathcal{E} \},
\]

and note that \( F \) is the feasible region of \( (\tilde{P}) \). We will prove that at each iteration the dimension of \( F \) decreases by one, and that each optimal solution of (1) lies in \( F \).

The next two lemmas show that at each iteration the dimension of \( F \) decreases by one. To prove it, in Lemma 4 below, we first show that at step 3 we always find a new index \( h \in \mathcal{F} \) that is not already in \( \mathcal{E} \).

Lemma 4. In step 3 of the iterative algorithm, we have \( \mathcal{F} \setminus \mathcal{E} \neq \emptyset \). In particular, the index \( h \) exists at each iteration.

Proof. Let \( \check{c}, \bar{c}, \check{x} \) and \( \check{y} \) be the vectors computed at a generic iteration of the iterative algorithm. Let \( \tilde{c} = \frac{n^2k\alpha}{\|\check{c}\|_\infty} \check{c} \), and note that \( \tilde{c} = [\hat{c}] \). Moreover, we have \( \|\hat{c}\|_\infty = n^3k\alpha \) and, since the largest absolute value of an entry of \( \hat{c} \) is the integer \( n^3k\alpha \), we also have \( \|\hat{c}\|_\infty = n^3k\alpha \).

Let \( B = \{ i \in \{1, \ldots, m\} \mid \bar{y}_i \neq 0 \} \). From property (i) of the vector \( \check{y} \) we know \( |B| \leq n \).

From the constraints of \( (\tilde{D}) \) we obtain

\[
\check{c} = \sum_{i \in [m]} a_i \check{y}_i = \sum_{i \in B} a_i \check{y}_i. \tag{2}
\]
Note that $\tilde{y}_j \geq 0$ for every $j \in B \setminus E$ since $\tilde{y}$ is feasible to $(\tilde{D})$. Hence to prove this lemma we only need to show that

$$|\tilde{y}_j| > nk \quad \text{for some } j \in B \setminus E.$$ (3)

The proof of (3) is divided into two cases.

In the first case we assume $B \cap E = \emptyset$. Thus, to prove (3), we only need to show that $|\tilde{y}_j| > nk$ for some $j \in B$. To obtain a contradiction, we suppose $|\tilde{y}_j| \leq nk$ for every $j \in B$. From (2) we obtain

$$\|\tilde{c}\|_\infty \leq \sum_{j \in B} |a_j\tilde{y}_j|_\infty = \sum_{j \in B} (|\tilde{y}_j| \|a_j\|_\infty) \leq \sum_{j \in B} (nk \cdot \alpha) \leq n^2 k \alpha.$$ However, this contradicts the fact that $\|\tilde{c}\|_\infty = n^3 k \alpha$. Thus $|\tilde{y}_j| > nk$ for some $j \in B$, and (3) holds. This concludes the proof in the first case.

In the second case we assume that $B \cap E$ is nonempty. In particular, we have $|B \setminus E| \leq n - 1$. In order to derive a contradiction, suppose that (3) does not hold, i.e., $|\tilde{y}_j| \leq nk$ for every $j \in B \setminus E$. From (2) we obtain

$$\tilde{c} = \sum_{i \in B} a_i\tilde{y}_i = \sum_{i \in B \cap E} a_i\tilde{y}_i + \sum_{j \in B \setminus E} a_j\tilde{y}_j.$$ Then

$$\left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\right\|_\infty \leq \sum_{j \in B \cap E} \|\tilde{y}_j\|_\infty \leq \sum_{j \in B \setminus E} (|\tilde{y}_j| \|a_j\|_\infty) \leq \sum_{j \in B \setminus E} (nk \cdot \alpha) \leq (n - 1) nk \alpha \leq n^2 k \alpha - 1.$$ (4)

To derive a contradiction, we prove the following claim.

Claim 1. We have $\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\|_\infty > n^2 k \alpha - 1$.

Proof of claim. By adding and removing $\tilde{c}$ inside the norm in the left-hand side below, we obtain

$$\left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\right\|_\infty = \left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i - (\tilde{c} - \tilde{c})\right\|_\infty \leq \left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\right\|_\infty + \|\tilde{c} - \tilde{c}\|_\infty.$$ (5)

Let us now focus on the left-hand side of (5). We have that $\tilde{c}$ is orthogonal to $a_i$, for every $i \in E$. This is because $\tilde{c}$ is a scaling of $\tilde{c}$ and the latter vector is, by definition, orthogonal to $a_i$, for every $i \in E$. We obtain

$$\left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\right\|_\infty \geq \frac{1}{\sqrt{n}} \left\|\tilde{c} - \sum_{i \in B \cap E} a_i\tilde{y}_i\right\|_2 \geq \frac{\|\tilde{c}\|_2}{\sqrt{n}} \geq \frac{\|\tilde{c}\|_\infty}{\sqrt{n}} = \frac{n^3 k \alpha}{\sqrt{n}} \geq n^2 k \alpha,$$ (6)

where the second inequality holds by Pitagora's theorem.
Using (5), (6), and noting that \( \|\tilde{c} - \hat{c}\|_\infty < 1 \) by definition of \( \tilde{c} \), we obtain

\[
\left\| \tilde{c} - \sum_{i \in B \cap E} a_i \tilde{y}_i \right\|_\infty \geq \left\| \hat{c} - \sum_{i \in B \cap E} a_i \tilde{y}_i \right\|_\infty - \|\tilde{c} - \hat{c}\|_\infty > n^2 k \alpha - 1.
\]

This concludes the proof of the claim. \( \diamond \)

Claim 1 and (4) yield a contradiction, implying that (3) holds. This concludes the proof in the second case.

The next lemma immediately implies that at each iteration the dimension of \( F \) decreases by 1. This will be used to prove that the algorithm performs at most \( n \) iterations.

**Lemma 5.** At each iteration, the row submatrix of \( A \) indexed by \( E \) has full row rank. Furthermore, at each iteration, its number of rows increases by exactly one.

**Proof.** We prove this lemma recursively. Clearly, the statement holds at the beginning of the algorithm because we have \( E = \emptyset \).

Assume now that, at a general iteration, the row submatrix of \( A \) indexed by \( E \) has full row rank. From Lemma 4, the index \( h \in F \setminus E \) defined in step 3 of the algorithm exists. From property (ii) of the vector \( \tilde{y} \), we have that \( a_h \) is linearly independent from the vectors \( a_i, i \in E \). Hence the rank of the row submatrix of \( A \) indexed by \( E \cup \{h\} \) is one more than the rank of the row submatrix of \( A \) indexed by \( E \). In particular, it has full row rank. \( \square \)

In the next three lemmas, we will prove that, at each iteration, every optimal solution to (1) lies in \( F \). Note that, since \( F \) is a face of \( P \), it is also a \([0,k]\)-polytope.

Suppose that an optimal solution \( \tilde{y} \) of (\( \tilde{D} \)) is known. The complementary slackness conditions for linear programming imply that, for every \( \tilde{x} \) optimal for (\( \tilde{P} \)):

\[
\tilde{y}_i > 0 \quad \Rightarrow \quad a_i^\top \tilde{x} = b_i \quad \quad i \in [m] \setminus E. \tag{7}
\]

Thus, in order to solve (\( \hat{P} \)), we can restrict the feasible region of (\( \hat{P} \)) by setting the primal constraints in (7) to equality.

Now, let (\( \hat{P} \)) and (\( \hat{D} \)) be the primal/dual pair obtained from (\( \hat{P} \)) and (\( \hat{D} \)) by replacing \( \hat{c} \) with a different vector \( \hat{c} \in \mathbb{R}^n \). To solve (\( \hat{P} \)), we wish to obtain a condition, similar to (7), that allows us to restrict its feasible region by exploiting knowledge of an optimal dual solution \( \tilde{y} \) of (\( \hat{D} \)). Namely, we show that if \( \tilde{y} \) is ‘close’ to being feasible for (\( \hat{D} \)), then for each index \( i \in [m] \setminus E \) such that \( \tilde{y}_i \) is sufficiently large, we have that the corresponding primal constraint is active at every optimal solution to (\( \hat{P} \)). Thus, in order to solve (\( \hat{P} \)), we can restrict the feasible region of (\( \hat{P} \)) by setting these primal constraints to equality.

In the following, for \( u \in \mathbb{R}^n \) we denote by \( |u| \) the vector whose entries are \( |u_i|, i = 1, \ldots n \).
Lemma 6. Let \( \hat{x} \in F, \hat{c} \in \mathbb{R}^n, \) and \( \hat{y} \in \mathbb{R}^m \) be such that
\[
\begin{align*}
|A^T \hat{y} - \hat{c}| &\leq 1 \\ 
\hat{y}_i &\geq 0 \quad i \in [m] \setminus \mathcal{E} \\ 
\hat{y}_i > 0 \quad &\Rightarrow \quad a_i^T \hat{x} = b_i \quad i \in [m] \setminus \mathcal{E}.
\end{align*}
\]
Then for any vector \( \hat{x} \in F \cap \mathbb{Z}^n \) with \( \hat{c}^T \hat{x} \geq \hat{c}^T \hat{x} \), we have
\[
\begin{align*}
\hat{y}_i > nk \quad &\Rightarrow \quad a_i^T \hat{x} = b_i \quad i \in [m] \setminus \mathcal{E}.
\end{align*}
\]
Proof. Let \( u := (\hat{x} - \hat{x}) \), and let \( u^+, u^- \in \mathbb{R}_+^n \) be defined as follows. For \( j \in [n] \),
\[
\begin{align*}
u_j^+ := \begin{cases} u_j & \text{if } u_j \geq 0 \\ 0 & \text{if } u_j < 0 \end{cases} \quad u_j^- := \begin{cases} 0 & \text{if } u_j \geq 0 \\ -u_j & \text{if } u_j < 0. \end{cases}
\end{align*}
\]

Clearly \( u = u^+ - u^- \) and \( |u| = u^+ + u^- \).

We prove this lemma by contradiction. Since \( \hat{c}^T \hat{x} \geq \hat{c}^T \hat{x} \), we have \( \hat{c}^T u \geq 0 \). Suppose that there exists \( h \in [m] \setminus \mathcal{E} \) such that \( \hat{y}_h > nk \) and \( a_h^T \hat{x} \neq b_h - 1 \). Since \( \hat{x} \in F \cap \mathbb{Z}^n \), we have \( a_h^T \hat{x} \leq b_h - 1 \). We rewrite (8) as \( A^T \hat{y} - 1 \leq \hat{c} \leq A^T \hat{y} + 1 \). Thus
\[
\begin{align*}
\hat{c}^T u &= \hat{c}^T u^+ - \hat{c}^T u^- \leq (A^T \hat{y} + 1)^T u^+ - (A^T \hat{y} - 1)^T u^- \\
&= (A^T \hat{y})^T (u^+ - u^-) + 1^T (u^+ + u^-) = (A^T \hat{y})^T u + 1^T |u|.
\end{align*}
\]
We can upper bound \( 1^T |u| \) in (12) by observing that \( |u_j| \leq k \) for all \( j \in [n] \), since \( u \) is the difference of two vectors in \([0, k]^n \). Thus
\[
1^T |u| \leq nk.
\]

We now compute an upper bound for \( (A^T \hat{y})^T u = \hat{y}^T Au \) in (12).
\[
\begin{align*}
\hat{y}^T Au &= \hat{y}_h a_h^T u + \sum_{i \in \mathcal{E}} \hat{y}_i a_i^T u + \sum_{i \in [m] \setminus \mathcal{E}, i \neq h} \hat{y}_i a_i^T u \\
&\leq -nk + \sum_{i \in \mathcal{E}} \hat{y}_i a_i^T u + \sum_{i \in [m] \setminus \mathcal{E}, i \neq h} \hat{y}_i a_i^T u \\
&\leq -nk + \sum_{i \in [m] \setminus \mathcal{E}, i \neq h} \hat{y}_i a_i^T u \\
&\leq -nk.
\end{align*}
\]

The strict inequality in (14) is implied by condition (10). In fact, \( \hat{y}_h > nk > 0 \), thus we have \( a_h^T \hat{x} = b_h \). Since \( a_h^T \hat{x} \leq b_h - 1 \), we get \( a_h^T u \leq -1 \). We multiply \( \hat{y}_h > nk \) by \( a_h^T u \) and obtain \( \hat{y}_h a_h^T u < nk \cdot a_h^T u \leq -nk \). Inequality (15) follows from the fact that, for each \( i \in \mathcal{E} \) we have \( a_i^T \hat{x} = b_i \) and \( a_i^T \hat{x} = b_i \) since both \( \hat{x} \) and \( \hat{x} \) are in \( F \), thus \( a_i^T u = 0 \). Inequality (16) follows from (9). To see why inequality (17) holds, first note that, from condition (10), \( \hat{y}_i > 0 \) implies \( a_i^T \hat{x} = b_i \). Furthermore, since \( \hat{x} \in F \), we have \( a_i^T \hat{x} \leq b_i \). Hence we have \( a_i^T u \leq 0 \) and \( \hat{y}_i a_i^T u \leq 0 \). By combining (12), (13) and (17) we obtain \( \hat{c}^T u < 0 \). This is a contradiction since we have previously seen that \( \hat{c}^T u \geq 0 \).
For a vector \( w \in \mathbb{Z}^n \) and a polyhedron \( Q \subseteq \mathbb{R}^n \), we say that a vector is \( w \)-maximal in \( Q \) if it maximizes \( w^\top x \) over \( Q \).

**Lemma 7.** The set \( F \) given at step 3 of the iterative algorithm is such that every vector \( \hat{x} \) that is \( \hat{c} \)-maximal in \( F \) satisfies \( a_i^\top \hat{x} = b_i \) for \( i \in F \).

**Proof.** Clearly, we just need to prove the lemma for every vertex \( \hat{x} \) of \( F \) that maximizes \( \overline{c}^\top x \) over \( F \). In particular, \( \hat{x} \) is a vertex of \( P \) and is therefore integral.

Define \( \hat{c} \in \mathbb{R}^n \) as \( \hat{c}_i := \frac{\alpha_i}{\|c\|_\infty} \overline{c}_i \) for \( i = 1, \ldots, n \). At step 3, \( \hat{x} \) is an optimal vertex of \((\hat{P})\), and \( \hat{y} \) is an optimal solution to the dual \((\hat{D})\). We have:

\[
A^\top \hat{y} - \hat{c} \leq 1 \\
\hat{y}_i \geq 0 \\
i \in [m] \setminus \mathcal{E}.
\]

Constraints (19) are satisfied since \( \hat{y} \) is feasible for \((\hat{D})\). Condition (18) holds because \( |A^\top \hat{y} - \hat{c}| = |\overline{c} - \hat{c}| = \overline{c} - \hat{c} < 1 \). Moreover, the complementary slackness conditions (7) are satisfied by \( \hat{x} \) and \( \hat{y} \), because they are optimal for \((\hat{P})\) and \((\hat{D})\), respectively.

Thus, \( A, b, \hat{c}, \hat{x}, \hat{y} \) satisfy the hypotheses of Lemma 6. Since the vector \( \hat{x} \) is \( \hat{c} \)-maximal in \( F \) and \( \hat{c} \) is a scaling of \( \overline{c} \), the vector \( \hat{x} \) is also \( \overline{c} \)-maximal in \( F \). Since \( \hat{x} \in F \), we have \( \overline{c}^\top \hat{x} \geq \overline{c}^\top \hat{x} \). Then Lemma 6 implies

\[
\hat{y}_i > nk \quad \Rightarrow \quad a_i^\top \hat{x} = b_i \\
i \in [m] \setminus \mathcal{E},
\]

that is, \( a_i^\top \hat{x} = b_i \) for all \( i \in \mathcal{F} \).

**Lemma 8.** The set \( \mathcal{E} \) updated in step 3 of the iterative algorithm is such that every vector \( x^* \) that is \( c \)-maximal in \( P \) satisfies \( a_i^\top x^* = b_i \) for \( i \in \mathcal{E} \).

**Proof.** Consider a vector \( x^* \) that is \( c \)-maximal in \( P \). We prove this lemma recursively. Clearly, the statement is true at the beginning of the algorithm, when \( \mathcal{E} = \emptyset \).

Suppose now that the statement is true at the beginning of a general iteration. At the beginning of step 3 we have that \( x^* \) is \( c \)-maximal in \( F \), thus it is also \( \overline{c} \)-maximal in \( F \). When we add an index \( h \in F \setminus \mathcal{E} \) to \( \mathcal{E} \) at the end of step 3, by Lemma 7 we obtain that \( a_h^\top x^* = b_h \). Thus, at each iteration of the algorithm we have \( a_i^\top x^* = b_i \) for \( i \in \mathcal{E} \).

The iterative algorithm ends if, at step (1), we have \( \overline{c} = 0 \). In the next lemma, we show that if this condition is satisfied, then the vector \( x^* \) returned by the algorithm solves (1).

**Proposition 2.** The vector \( x^* \) returned by the iterative algorithm at step 1 is an optimal solution to the LP problem (1).

**Proof.** Up to reordering the inequalities defining \( P \), we can assume, without loss of generality, that \( \mathcal{E} = \{1, \ldots, r\} \). Consider the following primal/dual pair:

\[
\begin{align*}
\max \quad & c^\top x \\
\text{s.t.} \quad & a_i^\top x = b_i \quad i = 1, \ldots, r \\
& a_i^\top x \leq b_i \quad i = r + 1, \ldots, m. \\
\end{align*}
\quad (P)
\quad \begin{align*}
\min \quad & b^\top y \\
\text{s.t.} \quad & A^\top y = c \\
& y_i \geq 0 \quad i = r + 1, \ldots, m.
\end{align*}
\quad (D)
\]

12
Note that the feasible region of \((P)\) is the face \(F\) of \(P\) obtained by setting to equality all the constraints indexed by \(E\), and the objective function of \((P)\) coincides with the one of \((1)\).

Let \(A_E\) be the row submatrix of \(A\) indexed by \(E\). By Lemma 5, the rank of \(A_E\) is \(r\). When, at step 1, we project \(c\) onto \(\{x \in \mathbb{R}^n \mid A_E x = 0\}\), we get \(\tilde{c} = c - A_E^\top (A_E A_E^\top)^{-1} A_E c\).

If \(\tilde{c} = 0\), we have \(c = A_E^\top z\), where \(z := (A_E A_E^\top)^{-1} A_E c\).

Let \(\bar{y} \in \mathbb{R}^m\) be defined by \(\bar{y}_i := z_i\) for \(i = 1, \ldots, r\), and \(\bar{y}_i := 0\) for \(i = r + 1, \ldots, m\).

First, \(\bar{y}\) is feasible for \((D)\). In fact \(\bar{y}_i \geq 0\) for \(i = r + 1, \ldots, m\) and

\[
A^\top \bar{y} = \sum_{i=1}^m \bar{y}_i a_i = \sum_{i=1}^r \bar{y}_i a_i = \sum_{i=1}^r z_i a_i = A_E^\top z = c.
\]

Consider now the vector \(x^*\) returned by the iterative algorithm at step 1. In particular, \(x^*\) is feasible for \((P)\). We have

\[
c^\top x^* = \bar{y}^\top Ax^* = \sum_{i=1}^r \bar{y}_i a_i^\top x^* + \sum_{i=r+1}^m \bar{y}_i a_i^\top x^* = \sum_{i=1}^r \bar{y}_i b_i = \bar{y}^\top b.
\]

By strong duality, \(x^*\) is \(c\)-maximal in \(F\). If \(x^*\) is not \(c\)-maximal in \(P\), then there exist a different vector \(x^\dagger\) that is \(c\)-maximal in \(P\). In particular, we have \(c^\top x^\dagger > c^\top x^*\). From Lemma 8, the vector \(x^\dagger\) lies in \(F\). Since \(x^*\) is \(c\)-maximal in \(F\), we obtain \(c^\top x^\dagger \leq c^\top x^*\), a contradiction. This shows that \(x^*\) is \(c\)-maximal in \(P\).

The next theorem is the main result of our paper.

**Theorem 1.** The length of the simplex path generated by the iterative algorithm is bounded by \(O(n^6 k \log k)\).

**Proof.** First, note that the iterative algorithm performs at most \(n\) iterations. This is because, by Lemma 5, at each iteration the rank of the row submatrix of \(A\) indexed by \(E\) increases by one. Therefore, at iteration \(n + 1\), the subspace \(\{x \in \mathbb{R}^n \mid a_i^\top x = 0 \text{ for } i \in E\}\) in step 1 is the origin. Hence the projection \(\tilde{c}\) of \(c\) onto this subspace is the origin, and the algorithm terminates by returning the current vector \(x^*\).

Each time the iterative algorithm performs step 3, it invokes the multi-stage algorithm with input \(F\), \(x^*\), and \(\tilde{c}\). Since \(F\) is a \([0, k]\)-polytope, by Proposition 1, each time the multi-stage algorithm is invoked, it generates a simplex path of length at most \(k n (\log \|\tilde{c}\|_\infty ) + 1\), where \(\|\tilde{c}\|_\infty = n^3 k \alpha\).

Denote by \(\varphi\) the facet complexity of \(P\) and by \(\nu\) the vertex complexity of \(P\). From Theorem 10.2 in [15], we know that facet complexity \(\varphi\) and the vertex complexity \(\nu\) of \(P\) are polynomially related, and in particular \(\varphi \leq 4 n^2 \nu\). Since \(P\) is a \([0, k]\)-polytope, we have \(\nu \leq n \log k\). Recall that each inequality in \(Ax \leq b\) is facet-defining and that the greatest common divisor of the entries in each row of \(A\) is one. Due to Remark 1.1 in [4], we obtain that \(\log \alpha \leq n \varphi\). Hence, we obtain \(\log \alpha \leq n \varphi \leq 4 n^3 \nu \leq 4 n^4 \log k\).

Thus, each time we run the multi-stage algorithm, we generate a simplex path of length in \(O(n^5 k \log k)\), and the simplex path generated throughout the entire algorithm has length in \(O(n^6 k \log k)\).

\[\square\]
We now bound the number of operations performed to construct the next vertex in the simplex path.

**Theorem 2.** The number of operations performed by the iterative algorithm to construct the next vertex in the simplex path is bounded by a polynomial in $m$, $n$ and $\log k$.

**Proof.** First, recall from the proof of Theorem 1 that the iterative algorithm performs at most $n$ iterations. Furthermore, we have $\log \alpha \leq 4n^4 \log k$, thus the size of $A$ is at most $nm \log \alpha \in O(mn^3 \log k)$. Moreover, each vector $\tilde{c}$ computed at step 2 is such that $\log \|\tilde{c}\|_{\infty} \in O(n^3 \log k)$.

We discuss the number of operations performed throughout the algorithm:

1. In step 1, computing the projection $\bar{c}$ of $c$ onto the subspace $\{x \in \mathbb{R}^n \mid a_i^T x = 0 \text{ for } i \in \mathcal{E}\}$ can be done in $O(n^3)$ steps via Gaussian elimination.

2. In step 2, computing the approximation $\tilde{c}$ of $\bar{c}$ can be done by binary search, and the number of comparisons required is at most $n \log \|\tilde{c}\|_{\infty} \in O(n^4 \log k)$.

3. In step 3, at the beginning of each execution of the basic algorithm within the multi-stage algorithm, we compute an approximation $\tilde{c}'$ of $\tilde{c}$. By construction, $\|\tilde{c}'\|_{\infty} \leq \|\tilde{c}\|_{\infty}$, thus the number of comparisons required is at most $n \log \|\tilde{c}\|_{\infty} \in O(n^4 \log k)$.

4. The number of operations performed by the oracle is bounded by a polynomial in the size of $A$, and hence it is bounded by a polynomial in $m$, $n$ and $\log k$.

5. At the end of step 3 we compute the vector $\tilde{y}$. From Lemma 3, the number of operations performed to compute this vector is bounded by a polynomial in the size of $A$, and hence it is bounded by a polynomial in $m$, $n$ and $\log k$.

Let $x^i$ be the $i$-th vertex of the simplex path computed by the iterative algorithm.

First, consider the case where $x^i$ is the optimal vertex returned by the multi-stage algorithm at step 3 in some iteration of the iterative algorithm. The algorithm then computes $\tilde{y}$ at the end of step 3, projects $c$ in step 1, computes $\tilde{c}$ in step 2, calculates an approximation $\tilde{c}'$ of $\tilde{c}$ at the beginning of the multi-stage algorithm in step 3, and finally calls once again the multi-stage algorithm to solve $(\tilde{P})$.

At this point, we distinguish two sub-cases. In the first sub-case, $x^i$ is not optimal for $(\tilde{P})$. Then, an oracle call at some iteration of the multi-stage algorithm yields the next vertex $x^{i+1}$ in the simplex path. Note that the multi-stage algorithm might call, in the worst case, $\log \|\tilde{c}\|_{\infty} \in O(n^3 \log k)$ times the basic algorithm before obtaining the next vertex $x^{i+1}$. Each time, the multi-stage algorithm first computes an approximation $\tilde{c}'$ of $\tilde{c}$ and then calls the basic algorithm which, in turn, invokes the oracle only once. Therefore, in this case, to compute $x^{i+1}$ we need a number of operations bounded a polynomial in $m$, $n$ and $\log k$.

In the second sub-case, $x^i$ is optimal for problem $(\tilde{P})$ solved by the multi-stage algorithm. In this case, a new full iteration of the iterative algorithm is performed. Since the iterative algorithm performs at most $n$ iterations, the next vertex $x^{i+1}$ in the simplex path requires the same number of operations as in the previous sub-case times $n$. Thus, it still performs a number of operations bounded a polynomial in $m$, $n$ and $\log k$. 

14
Finally, consider the case where $x^i$ is not the optimal vertex returned by multi-stage algorithm at step 3. Then the multi-stage algorithm might need to compute $\log \|\tilde{c}\|_\infty \in O(n^3 \log k)$ times an approximation $\tilde{c}^t$ of $\tilde{c}$ and to call the basic algorithm the same number of times. Every time, the oracle is called only once. Thus the runtime in this case is dominated by the runtime of the previous case.

References


