Discrete & Computational Geometry (2022) 67:503–524
https://doi.org/10.1007/s00454-020-00268-y

Short Simplex Paths in Lattice Polytopes

Alberto Del Pia[1] · Carla Michini[2]

Received: 21 March 2020 / Revised: 17 November 2020 / Accepted: 3 December 2020 /
Published online: 15 March 2021

Abstract
The goal of this paper is to design a simplex algorithm for linear programs on lattice polytopes that traces "short" simplex paths from any given vertex to an optimal one. We consider a lattice polytope P contained in $[0,k]^n$ and defined via m linear inequalities. Our first contribution is a simplex algorithm that reaches an optimal vertex by tracing a path along the edges of P of length in $O(n^4 k \log k)$. The length of this path is independent from m and it is the best possible up to a polynomial function. In fact, it is only polynomially far from the worst-case diameter, which roughly grows as nk. Motivated by the fact that most known lattice polytopes are defined via $0, \pm 1$ constraint matrices, our second contribution is a more sophisticated simplex algorithm which exploits the largest absolute value α of the entries in the constraint matrix. We show that the length of the simplex path generated by this algorithm is in $O(n^2 k \log (nk\alpha))$. In particular, if α is bounded by a polynomial in n, k, then the length of the simplex path is in $O(n^2 k \log (nk))$. For both algorithms, if P is "well described", then 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 in n and m, the algorithm runs in strongly polynomial time.

Keywords Lattice polytopes · Simplex algorithm · Diameter · Strongly polynomial time

Mathematics Subject Classification 90C05 · 52B20 · 52B05

Editor in Charge: Kenneth Clarkson

Alberto Del Pia
delpia@wisc.edu

Carla Michini
michini@wisc.edu

[1] Department of Industrial and Systems Engineering & Wisconsin Institute for Discovery, University of Wisconsin-Madison, Madison, WI, USA

[2] Department of Industrial and Systems Engineering, University of Wisconsin-Madison, Madison, WI, USA

1 Introduction

Linear programming (LP) is one of the most fundamental types of optimization models. In an 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

$$\max \{c^{\mathsf{T}} x \mid x \in P\}. \tag{1}$$

The polyhedron P is explicitly given via a system of linear inequalities, that is, $P = \{x \in \mathbb{R}^n \mid Ax \le b\}$, where $A \in \mathbb{Z}^{m\times n}$, $b \in \mathbb{Z}^m$. If P is nonempty and bounded, problem (1) admits an optimal solution that is a vertex of P.

In this paper, we consider the special class of LP problems (1) where P 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 to such optimization problems. A $[0,k]$-*polytope* in $\mathbb{R}^n$ is defined as a lattice polytope contained in the box $[0,k]^n$.

One of the main algorithms for LP is the simplex method. The simplex method moves from the current vertex to an adjacent one along an edge of the polyhedron P, until an optimal vertex x^* is reached or unboundedness is detected, 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*. The main objective of this paper is to design a simplex algorithm for $[0,k]$-polytopes that constructs "short" simplex paths from any starting vertex x^0.

But how short can a simplex path be? 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 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. It is known that the diameter of $[0,1]$-polytopes in $\mathbb{R}^n$ is at most n [23] and this bound is attained by the hypercube $[0,1]^n$. This upper bound was later generalized to nk for general $[0,k]$-polytopes in $\mathbb{R}^n$ [19], and refined to $\lfloor n(k-1/2)\rfloor$ for $k \ge 2$ [8] and to $nk - \lceil 2n/3\rceil - (k-3)$ for $k \ge 3$ [10]. For $k = 2$ the bound given in [8] is tight. In general, for fixed k, the diameter of lattice polytopes can grow linearly with n, since there are lattice polytopes, called primitive zonotopes, that can have diameter in $\Omega(n)$ [9,11]. Vice versa, when n is fixed, the diameter of a $[0,k]$-polytope in $\mathbb{R}^n$ can grow almost linearly with k. In fact, it is known that for $n = 2$ there are $[0,k]$-polytopes with diameter in $\Omega(k^{2/3})$ [1,4,29]. Moreover, for any fixed n, there are primitive zonotopes with diameter in $\Omega(k^{n/(n+1)})$ for k that goes to infinity [11,12].

Can we design a simplex algorithm whose simplex path length is only polynomially far from optimal, meaning that it is upper bounded by a polynomial function of the worst-case diameter? In this paper, we answer this question in the affirmative. Our first contribution is a preprocessing & scaling algorithm that generates a simplex path of length polynomially bounded in n and k, thus polynomially far from optimal.

Theorem 1.1 *The preprocessing & scaling algorithm generates a simplex path of length in* $O(n^4 k \log k)$.

We remark that the upper bound $O(n^4 k \log k)$. is independent from m. This is especially interesting because, even for $[0, 1]$-polytopes, m can grow exponentially in n (see, e.g., [26]).

The preprocessing & scaling algorithm is obtained by combining a preprocessing step by Frank and Tardos [15] with a bit scaling technique [2]. Namely, we first replace the cost vector c with a new cost vector $\breve{c}$ such that $\log \|\breve{c}\|$ is polynomially bounded in n and $\log k$. Next, we solve a sequence of LP problems where the cost vector is replaced with finer and finer integral approximations of $\breve{c}$, where each LP problem can be solved with any simplex algorithm.

Our next objective is that of decreasing the gap between the length $O(n^4 k \log k)$. provided by the preprocessing & scaling algorithm and the worst-case diameter, for wide classes of $[0, k]$-polytopes. We focus our attention on $[0, k]$-polytopes with bounded parameter α, defined as the largest absolute value of the entries in the constraint matrix A. This assumption is based on the fact that the overwhelming majority of $[0, k]$-polytopes arising in combinatorial optimization for which an external description is known, satisfies $\alpha = 1$ [26]. Our second contribution is a different simplex algorithm, named the face-fixing algorithm, which exploits the parameter α to significantly improve the dependence on n.

Theorem 1.2 *The face-fixing algorithm generates a simplex path of length in* $O(n^2 k \log (nk\alpha))$.

It should be noted that the dependence on α is only logarithmic, thus if α is bounded by a polynomial in n, k, then the length of our simplex path is in $O(n^2 k \log (nk))$. For $[0, 1]$-polytopes this bound reduces to $O(n^2 \log n)$.

At each iteration of the face-fixing algorithm we consider a face F of P containing all optimal solutions to (1). Then we compute a suitable approximation $\tilde{c}$ of c and we maximize $\tilde{c}^\mathsf{T} x$ over F. In order to solve this LP problem, we trace a path along the edges of P applying the same bit scaling technique introduced in our preprocessing & scaling algorithm. We also compute an optimal solution to the dual, which is used to identify a new constraint of $Ax \le b$ that is active at each optimal solution of (1). The face F of P is then updated for the next iteration by setting to equality such constraint, effectively restricting the feasible region to a lower dimensional polytope. The final simplex path is then obtained by merging together the different paths constructed at each iteration.

In both our simplex algorithms, under mild assumptions on the polytope P, 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, m, both our simplex algorithms are strongly polynomial. This assumption is justified by the existence of $[0, k]$-polytopes that, for fixed n, have a diameter that grows almost linearly in k [12]. 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 Sect. 2.4 in the book [25] for a thorough introduction.

2 The Preprocessing & Scaling Algorithm

In the remainder of the paper, we study problem (1) where P is a $[0,k]$-polytope.

All our algorithms are simplex algorithms, meaning that they explicitly construct a path along the edges of P from *any* given starting vertex x^0 to a vertex maximizing the linear function $c^\mathsf{T}x$. For this reason, we always assume that we are given a starting vertex x^0 of P. It should be noted that, if one is interested in obtaining an arbitrary starting vertex x^0, this can be accomplished via Tardos' algorithm by performing a number of operations that is polynomially bounded in size A. Recall that the *size* of the matrix A, denoted by size A, is in $O(nm\log\alpha)$ (see [25, Sect. 2.1] for more details). In a simplex algorithm one also expects that the next vertex in the simplex path can be computed in polynomial time. This is indeed the case for all our algorithms. In order to streamline the presentation, we defer these complexity issues to Sect. 2.4.

The ultimate goal of this section is to prove Theorem 1.1 by presenting and analyzing the preprocessing & scaling algorithm. Before doing that we need to introduce the basic algorithm in Sect. 2.1 and the scaling algorithm in Sect. 2.2.

Next we introduce our oracle, which provides a general way to construct the next vertex in the simplex path. In all our algorithms, the simplex path is constructed via a number of oracle calls with different inputs.

```
Oracle
Input: A polytope $P$, a cost vector $c \in \mathbb{Z}^n$, and a vertex $\bar{x}$ of $P$.
Output: Either a statement that $\bar{x}$ is optimal to (1), or a vertex adjacent to $\bar{x}$ with strictly larger cost.
```

We note that, whenever $\bar{x}$ is not optimal, our oracle has the freedom to return *any* adjacent vertex with strictly larger cost. Therefore, our algorithms can all be customized by further requiring the oracle to obey a specific pivoting rule.

2.1 The Basic Algorithm

The simplest way to solve (1) is to recursively invoke the oracle with input P, c, and the vertex obtained from the previous iteration, starting with the vertex x^0 in input. We formally describe this basic algorithm, which will be used as a subroutine in our subsequent algorithms.

```
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 $x^*$ of $P$ maximizing $c^\mathsf{T}x$.
  for $t = 0, 1, 2, \dots$ do
      Invoke oracle$(P, c, x^t)$.
      If the output of the oracle is a statement that $x^t$ is optimal, return $x^t$.
      Otherwise, set $x^{t+1} :=$ oracle$(P, c, x^t)$.
```

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

Observation 2.1 *The length of the simplex path generated by the basic algorithm is bounded by $c^{\mathsf{T}}x^* - c^{\mathsf{T}}x^0$. In particular, it is bounded by $nk\|c\|_\infty$.*

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

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

where, for the last inequality, we use $\left|x_i^* - x_i^0\right| \le k$ since P is a $[0,k]$-polytope. This concludes the proof of the second part of the statement. □

2.2 The Scaling Algorithm

The length of the simplex path generated by the basic algorithm is clearly not satisfactory. In fact, as we discussed in Sect. 1, our goal is to obtain a simplex path of length polynomial in n and k, and therefore independent from $\|c\|_\infty$. In this section we improve this gap by giving a scaling algorithm that yields a simplex path of length in $O(nk \log \|c\|_\infty)$.

Our scaling algorithm is based on a *bit scaling* technique. For ease of notation, we define $\ell := \lceil \log \|c\|_\infty \rceil$. The main idea is to iteratively use the basic algorithm with the sequence of increasingly accurate integral approximations of the cost vector c given by

$$c^t := \left\lceil \frac{c}{2^{\ell-t}} \right\rceil \qquad \text{for } t = 0, \dots, \ell.$$

Since c is an integral vector, we have $c^\ell = c$.

Bit scaling techniques have been extensively used since the 1970s to develop polynomial-time algorithms for a wide array of discrete optimization problems. Edmonds and Karp [14] and Dinic [13] independently introduced this technique in the context of the minimum cost flow problem. Gabow [16] used it for shortest path, maximum flow, assignment, and matching problems. The book [2] popularized bit scaling as a generic algorithmic tool in optimization. Bit scaling techniques have also been employed by Schulz et al. [27] (see also [20,24]) to design augmenting algorithms for 0/1-integer programming. To the best of our knowledge, bit scaling techniques have so far never been used to obtain short simplex paths in general lattice polytopes.

Next, we describe our algorithm.

Scaling 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 $x^*$ of $P$ maximizing $c^\mathsf{T} x$.
  for $t = 0, \dots, \ell$ do
      Compute $c^t$.
      Set $x^{t+1}$ := basic algorithm($P, c^t, x^t$).
  Return $x^{\ell+1}$.
```

The correctness of the scaling algorithm follows from the correctness of the basic algorithm, since the vector $x^{\ell+1}$ returned is the output of the basic algorithm with input P and cost vector $c^\ell = c$.

Next, we analyze the length of the simplex path generated by the scaling algorithm. The next two lemmas provide simple properties of the approximations c^t of c and are based on the common techniques used in most bit scaling algorithms.

Lemma 2.2 *For each $t = 0, \dots, \ell$, we have $\|c^t\|_\infty \le 2^t$.*

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

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

Lemma 2.3 *For each $t = 1, \dots, \ell$, we have $2c^{t-1} - c^t \in \{0, 1\}^n$.*

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

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

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

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

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

Proposition 2.4 *The length of the simplex path generated by the scaling algorithm is bounded by $nk\big(\lceil \log \|c\|_\infty \rceil + 1\big) \in O\big(nk \log \|c\|_\infty\big)$.*

Proof Note that the scaling 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 nk.

First we consider the iteration $t = 0$ of the scaling algorithm. In this iteration, the basic algorithm is called with input P, c^0, and x^0. Lemma 2.2 implies that $\|c^0\|_\infty \le 1$, and from Observation 2.1 we have that the basic algorithm calls the oracle at most nk times.

Next, consider the iteration t of the scaling algorithm for $t \in \{1, \ldots, \ell\}$. In this iteration, the basic algorithm is called with input P, c^t, and x^t, and outputs the vertex x^{t+1}. From Observation 2.1, we only need to show that ${c^t}^\mathsf{T} x^{t+1} - {c^t}^\mathsf{T} x^t \le nk$.

First, we derive an upper bound on ${c^t}^\mathsf{T} x^{t+1}$. From Lemma 2.3, the vector $2c^{t-1} - c^t$ is nonnegative. Since also x^{t+1} is nonnegative, we obtain ${c^t}^\mathsf{T} x^{t+1} \le 2{c^{t-1}}^\mathsf{T} x^{t+1}$. Furthermore, by definition of x^t we obtain ${c^{t-1}}^\mathsf{T} x^{t+1} \le {c^{t-1}}^\mathsf{T} x^t$. We thereby obtain the upper bound ${c^t}^\mathsf{T} x^{t+1} \le 2{c^{t-1}}^\mathsf{T} x^{t+1} \le 2{c^{t-1}}^\mathsf{T} x^t$.

We can now show ${c^t}^\mathsf{T} x^{t+1} - {c^t}^\mathsf{T} x^t \le nk$. We have

$$
{c^t}^\mathsf{T} x^{t+1} - {c^t}^\mathsf{T} x^t \le 2{c^{t-1}}^\mathsf{T} x^t - {c^t}^\mathsf{T} x^t = (2c^{t-1} - c^t)^\mathsf{T} x^t \le nk.
$$

The last inequality holds because, from Lemma 2.3, we know that each component of $2c^{t-1} - c^t$ is at most one, while the vector x^t is in $[0, k]^n$. $\square$

In the next section we use the scaling algorithm as a subroutine in the preprocessing & scaling algorithm. We remark that, the scaling algorithm will also be a subroutine in the face-fixing algorithm, which is described in Sect. 3.

2.3 The Preprocessing & Scaling Algorithm

The length of the simplex path generated by the scaling algorithm still depends on $\|c\|_\infty$, even though the dependence is now logarithmic instead of linear. In this section we show that we can completely remove the dependence on $\|c\|_\infty$ by using our scaling algorithm in conjunction with the preprocessing algorithm by Frank and Tardos [15]. This method relies on the simultaneous approximation algorithm of Lenstra et al. [21]. Next, we state the input and output of Frank and Tardos' algorithm.

Preprocessing algorithm

Input: A vector $c \in \mathbb{Q}^n$ and a positive integer N.

Output: A vector $\check{c} \in \mathbb{Z}^n$ such that $\|\check{c}\|_\infty \le 2^{4n^3} N^{n(n+2)}$ and $\operatorname{sign}(c^\mathsf{T} z) = \operatorname{sign}(\check{c}^\mathsf{T} z)$ for every $z \in \mathbb{Z}^n$ with $\|z\|_1 \le N - 1$.

The number of operations performed by the preprocessing algorithm is polynomially bounded in n and $\log N$. For more details, we refer the reader to [15, Sect. 3].

Next, we describe the algorithm obtained by combining the preprocessing algorithm and the scaling algorithm.

```
Preprocessing & scaling 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 $x^*$ of $P$ maximizing $c^{\mathsf{T}}x$.
  Set $\breve{c}$ := preprocessing algorithm$(c, N := nk+1)$.
  Set $x^*$ := scaling algorithm$(P, \breve{c}, x^0)$.
  Return $x^*$.
```

It is simple to see that the preprocessing & scaling algorithm is correct. In fact, due to the correctness of the scaling algorithm, we have that $\breve{c}^{\mathsf{T}}(x^* - x) \geq 0$ for every $x \in P$. Note that, for every $x \in P \cap \mathbb{Z}^n$, we have $x^* - x \in \mathbb{Z}^n$ and $\|x^* - x\|_1 \leq nk = N - 1$. Therefore, the preprocessing algorithm guarantees that $c^{\mathsf{T}}(x^* - x) \geq 0$ for every $x \in P \cap \mathbb{Z}^n$. The correctness of the preprocessing & scaling algorithm then follows because all vertices of P are integral.

We are now ready to give a proof of Theorem 1.1. We show that the obtained simplex path length is polynomially bounded in n and k, thus only polynomially far from the worst-case diameter.

Proof of Theorem 1.1 The vector $\breve{c}$ returned by the preprocessing algorithm satisfies $\|\breve{c}\|_\infty \leq 2^{4n^3}(nk+1)^{n(n+2)}$, hence $\log \|\breve{c}\|_\infty \leq 4n^3 + n(n+2)\log(nk+1)$. From Proposition 2.4, the length of the simplex path generated by the preprocessing & scaling algorithm is bounded by

$$nk\big(\lceil \log \|\breve{c}\|_\infty \rceil + 1\big) \leq nk\big(4n^3 + n(n+2)\log(nk+1) + 2\big) \in O(n^4 k \log k). \quad \square$$

Next, we compare our bound on the length of the simplex path constructed by the preprocessing & scaling algorithm with other known bounds for some classic pivoting rules that can be applied to $[0,k]$-polytopes.

A result by Kitahara and Mizuno [17,18] implies that, by using the dual simplex algorithm with Dantzig's or the best improvement pivoting rule, we can construct a simplex path in P whose length is at most $n^2 K \log(nK)$, where $K = \max\{k, S\}$ and $S = \max\{\|b - Ax\|_\infty \mid x \in P\}$. This has been recently extended to the steepest edge pivoting rule by Blanchard et al. [6]. We remark that K critically depends on the values of the slack variables of the constraints $Ax \leq b$. It is known that, even for $k = 1$, the value S can be as large as $(n-1)^{(n-1)/2}/2^{2n+o(n)}$ (see [3,30]), which is not polynomially bounded in n, k. As a consequence, this upper bound on the simplex path length is not polynomially bounded in n, k.

2.4 Complexity

The goal of this section is to analyze the number of operations performed by the preprocessing & scaling algorithm in order to construct the next vertex in the simplex path or to certify optimality of the current vertex. In particular we show that this number of operations is polynomially bounded in size A and $\log k$.

We start by analyzing the complexity of the basic algorithm. We recall that a d-dimensional polytope is said to be *simple* if each vertex is adjacent to exactly d other vertices. Throughout this paper we assume that a simple polytope is given by a minimal system of linear equalities defining its affine hull and a system of linear inequalities defining its facets.

Lemma 2.5 *The number of operations performed by the basic algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in* size A. *If P is simple, the number of operations is $O(nm)$.*

Proof To prove this proposition it suffices to show the following statement: an oracle call can be performed with a number of operations polynomially bounded in size A. If P is simple, it can be performed in $O(nm)$ operations.

First, consider the case where P is simple. We can then use a pivot of the dual simplex method, where the primal is in standard form, and the feasible region of the dual is given by the polytope P. This requires $O(nm)$ operations [5].

Consider now the case where P may not be simple. Denote by $A^= x \leq b^=$ the subsystem of the inequalities of $Ax \leq b$ satisfied with equality by $\bar{x}$. Note that the polyhedron $T := \{x \in \mathbb{R}^n \mid A^= x \leq b^=\}$ is a cone pointed at $\bar{x}$. Denote by d^T the sum of all the rows in $A^=$ and note that the vertex $\bar{x}$ is the unique maximizer of $d^\mathsf{T} x$ over T. Let T' be the truncated cone $T' := \{x \in T \mid d^\mathsf{T} x \geq d^\mathsf{T} \bar{x} - 1\}$ and note that there is a bijection between the vertices of P adjacent to $\bar{x}$ and the vertices of T' different from $\bar{x}$. We solve the LP problem $\max\{c^\mathsf{T} x \mid x \in T'\}$. Using Tardos' algorithm, this LP problem can be solved in a number of operations that is polynomial in the size of the constraint matrix, which is polynomial in size A.

If $\bar{x}$ is an optimal solution to the LP, then the oracle returns that $\bar{x}$ is optimal. Otherwise, Tardos' algorithm returns an optimal solution that is a vertex z of T' different from $\bar{x}$. In this case the oracle needs to return the corresponding adjacent vertex $\bar{z}$ of $\bar{x}$ in P. Let $A'x \leq b'$ be the system obtained from $Ax \leq b$ by setting to equality the inequalities in the subsystem $A^= x \leq b^=$ satisfied with equality by both $\bar{x}$ and z. It should be noted that the vectors that satisfy $A'x \leq b'$ constitute the edge of P between $\bar{x}$ and $\bar{z}$. The vector $\bar{z}$ can then be found by maximizing $c^\mathsf{T} x$ over $A'x \leq b'$ with Tardos' algorithm. □

Next we analyze the number of operations performed by the scaling algorithm.

Lemma 2.6 *The number of operations performed by the scaling algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in* size A *and* $\log \|c\|_\infty$. *If P is simple, the number of operations is in $O(nm \log^2 \|c\|_\infty)$.*

Proof To construct the next vertex in the simplex path or to certify optimality of the current vertex, in the worst case the scaling algorithm calls $\ell + 1$ times the basic algorithm. This happens if the first ℓ calls of the basic algorithm all return the current vertex. In each iteration the scaling algorithm first computes an approximation c^t of c and then calls the basic algorithm. Computing c^t can be done by binary search, and the number of comparisons required is at most $n \log \|c^t\|_\infty$, which is bounded by $nt \leq n\ell$ from Lemma 2.2. Furthermore, from Lemma 2.5, each time the basic algorithm is called, it performs a number of operations polynomially bounded in size A, and by $O(nm)$ if P is simple. Therefore the scaling algorithm performs a number of operations bounded by a polynomial in size A and in $\log \|c\|_\infty$. If P is simple, the number of operations is $O((\ell+1)(n\ell + O(nm))) = O(nm \log^2 \|c\|_\infty)$. □

We are now ready to analyze the complexity of the preprocessing & scaling algorithm.

Proposition 2.7 *The number of operations performed by the preprocessing & scaling algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in* size A *and* $\log k$*. If P is simple, the number of operations is polynomially bounded in n, m, and* $\log k$.

Proof The number of operations performed by the preprocessing & scaling algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is the sum of: (i) the number of operations needed to compute $\breve{c}$, and (ii) the number of operations performed by the scaling algorithm, with cost vector $\breve{c}$, to construct the next vertex in the simplex path or to certify optimality of the current vertex. The vector $\breve{c}$ can be computed with a number of operations polynomially bounded in n and $\log(nk)$ [15]. From Lemma 2.6, (ii) is polynomially bounded in size A and $\log \|\breve{c}\|_\infty$, and by $O(nm \log^2 \|\breve{c}\|_\infty)$ if P is simple. To conclude the proof, we only need to observe that $\log \|\breve{c}\|_\infty$ is polynomially bounded in n and $\log k$. □

3 The Face-Fixing Algorithm

In this section, our task is to construct a simplex path shorter than the one computed by the preprocessing & scaling algorithm, provided that the constraint matrix A has entries with small absolute value. We define $[m] := \{1, 2, \dots, m\}$ and refer to the rows of A as a_i^T, for $i \in [m]$. We recall that the parameter α denotes the largest absolute value of the entries in A. Next, we state the algorithm that will be analyzed in this section.

Face-fixing 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 $x^*$ of $P$ maximizing $c^\mathsf{T} x$.
0: Let $\mathcal{E} := \emptyset$ and $x^* := x^0$.
1: Let $\bar{c}$ be the projection of $c$ onto the subspace $\{x \in \mathbb{R}^n \mid a_i^\mathsf{T} x = 0 \text{ for } i \in \mathcal{E}\}$ of $\mathbb{R}^n$. If $\bar{c} = 0$ return $x^*$,
   otherwise go to 2.
2: Let $\hat{c} := (n^3 k\alpha / \|\bar{c}\|_\infty)\bar{c}$ and define $\tilde{c} \in \mathbb{Z}^n$ as $\tilde{c}_i := \lfloor \hat{c}_i \rfloor$ for $i = 1, \dots, n$.
3: Consider the following pair of primal and dual LP problems:
```

$$\begin{array}{lll} \max & \tilde{c}^\mathsf{T} x & \\ \text{s.t.} & a_i^\mathsf{T} x = b_i & \text{for } i \in \mathcal{E} \\ & a_i^\mathsf{T} x \le b_i & \text{for } i \in [m] \setminus \mathcal{E} \end{array} \qquad (\tilde{\mathrm{P}}) \qquad \begin{array}{lll} \min & b^\mathsf{T} y & \\ \text{s.t.} & \text{for } A^\mathsf{T} y = \tilde{c} & \\ & y_i \ge 0 & \text{for } i \in [m] \setminus \mathcal{E}. \end{array} \qquad (\tilde{\mathrm{D}})$$

```
   Use the scaling algorithm to compute an optimal vertex $\tilde{x}$ of ($\tilde{\mathrm{P}}$) starting from $x^*$.
   Compute an optimal solution $\tilde{y}$ to the dual ($\tilde{\mathrm{D}}$) 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 $\mathcal{H} := \{i \mid \tilde{y}_i > nk\}$. Update $\mathcal{E} := \mathcal{E} \cup \mathcal{H}$, $x^* := \tilde{x}$ and go back to step 1.
```

The above algorithm is iterative in nature. Precisely, an *iteration* of the algorithm corresponds to one execution of steps 1, 2, and 3 . The key idea is to identify, at each iteration, at least 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.

Throughout the algorithm, $\mathcal{E}$ contains the indices of the constraints that have been set to equality so far. Correspondingly, at each iteration, we restrict the feasible region to the face F of P defined as

$$F := \{x \in \mathbb{R}^n \mid a_i^\mathsf{T} x \leq b_i \text{ for } i \in [m] \setminus \mathcal{E} \text{ and } a_i^\mathsf{T} x = b_i \text{ for } i \in \mathcal{E}\}. \tag{2}$$

Note that, since F is a face of P, it is also a $[0, k]$-polytope. We also store and update at each iteration a vertex x^* of F. The vertex x^* is initially set to x^0 and it ends up being an optimal solution of (1).

In step 1, we compute the projection $\bar{c}$ of c onto $\{x \in \mathbb{R}^n \mid a_i^\mathsf{T} x = 0 \text{ for } i \in \mathcal{E}\}$. Maximizing $c^\mathsf{T} x$ over F is equivalent to maximizing $\bar{c}^\mathsf{T} x$ over the same set. Therefore, if $\bar{c} = 0$, then c is perpendicular to F and all the points in F are optimal. In this case the algorithm terminates by returning x^*.

In step 2, we first compute a scaling $\hat{c}$ of $\bar{c}$, and then an integer approximation $\tilde{c}$ of $\hat{c}$ such that $\|\tilde{c}\|_\infty = n^3 k\alpha$. This will be the key to obtain a short simplex path.

In step 3, we solve problem ($\tilde{\text{P}}$) and its dual. Note that ($\tilde{\text{P}}$) differs from (1) in two ways: first, the feasible region of ($\tilde{\text{P}}$) is the current face F of P. Second, the cost vector defining the objective function of ($\tilde{\text{P}}$) is the approximation $\tilde{c}$ of $\hat{c}$ computed in step 2. In order to solve ($\tilde{\text{P}}$), we apply the scaling algorithm from x^*, and we trace a path along the edges of F from x^* to an optimal vertex $\tilde{x}$ of ($\tilde{\text{P}}$). We also compute an optimal solution to the dual problem ($\tilde{\text{D}}$), which is used to identify a subset $\mathcal{H}$ of constraints of $Ax \leq b$ that are active at each optimal solution of (1). Finally, we add the indices of $\mathcal{H}$ to $\mathcal{E}$ and we set $x^* := \tilde{x}$.

At the end, the final simplex path from the input vertex x^0 to an optimal vertex of (1) is obtained by merging together the different paths constructed by the scaling algorithm at each iteration.

We remark that the face-fixing algorithm is well defined, meaning that it can indeed perform all the instructions stated in its steps. In particular, in step 3: (a) x^* is a valid input for the scaling algorithm and (b) a vector $\tilde{y}$ satisfying properties (i) and (ii) always exists. To see (a) we need to show that the vector x^* is in F. Consider the vectors $\tilde{x}$, $\tilde{y}$ and an index $h \in \mathcal{H}$ from the previous iteration, and note that $\tilde{x} = x^*$. We have $\tilde{y}_h > 0$, which, by complementary slackness, implies $a_h^\mathsf{T} \tilde{x} = b_h$. This immediately implies that x^* is indeed feasible for the problem ($\tilde{\text{P}}$) of the current iteration. The proof of (b) is more technical and it involves some standard LP arguments. We defer the proof of (b) to Sect. 3.3 (see Lemma 3.6).

Our face-fixing algorithm is inspired by Tardos' strongly polynomial algorithm for combinatorial problems [28]. Tardos' algorithm solves an LP problem in standard form in a number of operations bounded by a polynomial in the size of the constraint matrix. The main similarity between the two algorithms is that both recursively restrict the feasible region to a lower dimensional face of the feasible region. The three main differences between the face-fixing algorithm and Tardos' algorithm are: (1) Tardos' algorithm solves LP problems in standard form, while we consider a polytope P in general form, i.e., $P = \{x \in \mathbb{R}^n \mid Ax \le b\}$; (2) Tardos' algorithm exploits as a parameter the largest absolute value of a subdeterminant of the constraint matrix, while our algorithm relies on parameters k and α; and (3) Tardos' algorithm is *not* a simplex algorithm, while ours is, since it traces a simplex path along the edges of P.

3.1 Correctness

Our first goal is to prove that the face-fixing algorithm is correct. First, we will prove that the algorithm terminates in a finite number of iterations. Next, we will prove that the algorithm returns a vertex of P maximizing $c^{\mathsf{T}}x$. Recall that, at each iteration, we are restricting our search to the face F of P defined by (2), which is obtained by setting to equality the constraints indexed by $\mathcal{E}$.

To prove that the algorithm terminates in a finite number of iterations, we will show that at the end of iteration j the dimension of F is at most $n-j$. This will immediately imply that face-fixing algorithm performs at most $n+1$ iterations. To prove this, we will show that at step 3 we always add at least an index to $\mathcal{E}$. Equivalently, we will prove that there is an index $h \in [m] \setminus \mathcal{E}$ such that $\tilde{y}_h > nk$. The key idea of the proof consists in relating the complexity of $\tilde{c}$ to the complexity of $\tilde{y}$ by exploiting the dual constraints. If no dual variable is larger than the prescribed threshold of nk, then the infinity norm of $\tilde{c}$ is too small, and we obtain a contradiction.

Lemma 3.1 *The face-fixing algorithm performs at most* $n+1$ *iterations.*

Proof It suffices to show that in step 3 of the face-fixing algorithm we have $\mathcal{H} \setminus \mathcal{E} \neq \emptyset$ at each iteration. In fact, this implies that there exists an index $h \in \mathcal{H} \setminus \mathcal{E}$. In particular, $\tilde{y}_h > nk$ and from property (ii) of the vector $\tilde{y}$, we have that a_h is linearly independent from the vectors a_i, $i \in \mathcal{E}$. Hence, at each iteration, the rank of the row submatrix of A indexed by $\mathcal{E}$ increases by at least one. Therefore, after at most n iterations, the subspace $\{x \in \mathbb{R}^n \mid a_i^{\mathsf{T}}x = 0 \text{ for } i \in \mathcal{E}\}$ in step 1 is the origin. Hence the projection $\bar{c}$ of c onto this subspace is the origin, and the algorithm terminates by returning the current vector x^*. Therefore, in the remainder of the proof we show that in step 3 of the face-fixing algorithm, we have $\mathcal{H} \setminus \mathcal{E} \neq \emptyset$ at each iteration.

Let $\bar{c}$, $\hat{c}$, $\tilde{c}$, $\tilde{x}$, and $\tilde{y}$ be the vectors computed at a generic iteration of the face-fixing algorithm. Recall that $\tilde{c} = \lfloor \hat{c} \rfloor$. Moreover, we have $\|\hat{c}\|_\infty = n^3 k\alpha$ and, since this number is an integer, we also have $\|\tilde{c}\|_\infty = n^3 k\alpha$.

Let $\mathcal{B} = \{i \in \{1, \dots, m\} \mid \tilde{y}_i \neq 0\}$. From property (i) of the vector $\tilde{y}$ we know $|\mathcal{B}| \le n$. From the constraints of ($\tilde{\mathrm{D}}$) and from the definition of $\mathcal{B}$ we obtain

$$\tilde{c} = \sum_{i \in [m]} a_i \tilde{y}_i = \sum_{i \in \mathcal{B}} a_i \tilde{y}_i. \tag{3}$$

Note that $\tilde{y}_j \geq 0$ for every $j \in \mathcal{B} \setminus \mathcal{E}$ since $\tilde{y}$ is feasible to ($\tilde{\mathrm{D}}$). Hence to prove this lemma we only need to show that

$$\left|\tilde{y}_j\right| > nk \qquad \text{for some } \; j \in \mathcal{B} \setminus \mathcal{E}. \tag{4}$$

The proof of (4) is divided into two cases.

In the first case we assume $\mathcal{B} \cap \mathcal{E} = \emptyset$. Thus, to prove (4), we only need to show that $|\tilde{y}_j| > nk$ for some $j \in \mathcal{B}$. To obtain a contradiction, we suppose $|\tilde{y}_j| \leq nk$ for every $j \in \mathcal{B}$. From (3) we obtain

$$\|\tilde{c}\|_\infty \leq \sum_{j \in \mathcal{B}} \|a_j \tilde{y}_j\|_\infty = \sum_{j \in \mathcal{B}} (|\tilde{y}_j| \cdot \|a_j\|_\infty) \leq \sum_{j \in \mathcal{B}} (nk \cdot \alpha) \leq n^2 k\alpha.$$

However, this contradicts the fact that $\|\tilde{c}\|_\infty = n^3 k\alpha$. Thus $\left|\tilde{y}_j\right| > nk$ for some $j \in \mathcal{B}$, and (4) holds. This concludes the proof in the first case.

In the second case we assume that $\mathcal{B} \cap \mathcal{E}$ is nonempty. In particular, we have $|\mathcal{B} \setminus \mathcal{E}| \leq n-1$. In order to derive a contradiction, suppose that (4) does not hold, i.e., $\left|\tilde{y}_j\right| \leq nk$ for every $j \in \mathcal{B} \setminus \mathcal{E}$. From (3) we obtain

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

Then

$$\begin{aligned} \left\| \tilde{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_\infty &\leq \sum_{j \in \mathcal{B} \setminus \mathcal{E}} \|a_j \tilde{y}_j\|_\infty = \sum_{j \in \mathcal{B} \setminus \mathcal{E}} (|\tilde{y}_j| \cdot \|a_j\|_\infty) \\ &\leq \sum_{j \in \mathcal{B} \setminus \mathcal{E}} (nk \cdot \alpha) \leq (n-1)nk\alpha \leq n^2 k\alpha - 1. \end{aligned} \tag{5}$$

Next, in order to derive a contradiction, we show that

$$\left\| \tilde{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_\infty > n^2 k\alpha - 1. \tag{6}$$

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

$$\begin{aligned} \left\| \hat{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_\infty &= \left\| \tilde{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i - (\tilde{c} - \hat{c}) \right\|_\infty \\ &\leq \left\| \tilde{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_\infty + \|\tilde{c} - \hat{c}\|_\infty. \end{aligned} \tag{7}$$

Let us now focus on the left-hand side of (7). We have that $\hat{c}$ is orthogonal to a_i, for every $i \in \mathcal{E}$. This is because $\hat{c}$ is a scaling of $\bar{c}$ and the latter vector is, by definition, orthogonal to a_i, for every $i \in \mathcal{E}$. We obtain

$$\left\| \hat{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_\infty \geq \frac{1}{\sqrt{n}} \left\| \hat{c} - \sum_{i \in \mathcal{B} \cap \mathcal{E}} a_i \tilde{y}_i \right\|_2 \geq \frac{\|\hat{c}\|_2}{\sqrt{n}} \geq \frac{\|\hat{c}\|_\infty}{\sqrt{n}} = \frac{n^3 k\alpha}{\sqrt{n}} \geq n^2 k\alpha, \tag{8}$$

where the second inequality holds by Pythagoras' theorem. Using (7), (8), and noting that $\|\tilde{c} - \hat{c}\|_\infty < 1$ by definition of $\tilde{c}$, we obtain

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

This concludes the proof of (6). Inequalities (5) and (6) yield a contradiction, thus (4) holds. This concludes the proof in the second case. □

Lemma 3.1 relies on proving that at step 3 we always add at least an index to $\mathcal{E}$. This is reminiscent of [28, Lem. 1.2]. The first key difference is that in our setting the primal is in general form, while in [28] the primal is in standard form. As a consequence, the proof of [28, Lem. 1.2] is a bit more direct, while in our setting the proof is more technical. In particular, as explained above, a crucial step of our proof consists in using the dual constraints to relate the infinity norm of $\tilde{c}$ to the infinity norm of $\tilde{y}$. In this step, we exploit the bound of α on the largest absolute value of an entry of A. The second key difference is that the only parameter that is used in [28, Lem. 1.2] is the largest absolute value of a subdeterminant of the constraint matrix, while Lemma 3.1 relies on the two parameters k and α.

In the next two lemmas we will prove that, at each iteration, every optimal solution to (1) lies in F. This is trivially true at the beginning of the algorithm, when $\mathcal{E} = \emptyset$. Thus, we will need to prove that when we update $\mathcal{E}$ in step 3 the property remains valid with respect to the new face of P defined by $\mathcal{E}$. Note that, to update $\mathcal{E}$ in step 3, we use an optimal solution $\tilde{y}$ of the dual ($\tilde{\mathrm{D}}$) with right-hand-side $\tilde{c}$. However, our ultimate goal is to achieve optimality with respect to c, or equivalently to $\hat{c}$. In other words, we would like to solve the problem ($\hat{\mathrm{P}}$) obtained from ($\tilde{\mathrm{P}}$) by replacing $\tilde{c}$ with $\hat{c}$. How can we exploit $\tilde{y}$ to detect a new constraint that is satisfied with equality by all optimal solutions to ($\hat{\mathrm{P}}$)?

The standard complementary slackness conditions for LP state that, for each positive component of $\tilde{y}$, the corresponding constraint of ($\tilde{\mathrm{P}}$) must be satisfied with equality by all the optimal solutions to ($\tilde{\mathrm{P}}$). In Lemma 3.2, we define a variant of these conditions that relate the problems ($\hat{\mathrm{P}}$) and ($\tilde{\mathrm{D}}$). We will obtain that, for each component of $\tilde{y}$ that is above a prescribed threshold, the corresponding constraint of ($\hat{\mathrm{P}}$) must be satisfied with equality by all the optimal solutions to ($\hat{\mathrm{P}}$). The key idea of the proof is to exploit the fact that the cost vector $\hat{c}$ in ($\hat{\mathrm{P}}$) and the right-hand-side vector $\tilde{c}$ in ($\tilde{\mathrm{D}}$) are not very far apart, since by definition $\tilde{c} = \lfloor \hat{c} \rfloor$. Thus, the optimal solution $\tilde{y}$ of ($\tilde{\mathrm{D}}$) is "almost feasible" for the dual problem associated to ($\hat{\mathrm{P}}$).

In the following, $\mathbf{1}$ denotes the vector of all ones, and for $u \in \mathbb{R}^n$ we denote by $|u|$ the vector whose entries are $|u_i|$, $i = 1, \ldots n$.

Lemma 3.2 *Let $\tilde{x}$ and $\tilde{y}$ be the vectors computed at step 3 of the face-fixing algorithm, and denote by F the feasible set of ($\tilde{\mathrm{P}}$). Then for any vector $\hat{x} \in F \cap \mathbb{Z}^n$ such that $\hat{c}^{\mathsf{T}}\hat{x} \geq \hat{c}^{\mathsf{T}}\tilde{x}$, we have*

$$\tilde{y}_i > nk \quad \Rightarrow \quad a_i^{\mathsf{T}}\hat{x} = b_i, \qquad i \in [m] \setminus \mathcal{E}. \tag{9}$$

Proof First, since $\tilde{y}$ is feasible for ($\tilde{\mathrm{D}}$), we have $|A^{\mathsf{T}}\tilde{y} - \hat{c}| = |\tilde{c} - \hat{c}| = \hat{c} - \tilde{c}$. Since $\tilde{c} = \lfloor \hat{c} \rfloor$ we obtain

$$|A^{\mathsf{T}}\tilde{y} - \hat{c}| \leq \mathbf{1}. \tag{10}$$

Moreover, since $\tilde{x}$ and $\tilde{y}$ are optimal for ($\tilde{\mathrm{P}}$) and ($\tilde{\mathrm{D}}$), respectively, they satisfy the complementary slackness conditions

$$\tilde{y}_i > 0 \quad \Rightarrow \quad a_i^{\mathsf{T}}\tilde{x} = b_i, \qquad i \in [m] \setminus \mathcal{E}. \tag{11}$$

Let $u := \hat{x} - \tilde{x}$, and let $u^+, u^- \in \mathbb{R}^n_+$ be defined as follows. For $j \in [n]$,

$$u_j^+ := \begin{cases} u_j & \text{if } u_j \geq 0, \\ 0 & \text{if } u_j < 0, \end{cases} \qquad u_j^- := \begin{cases} 0 & \text{if } u_j \geq 0, \\ -u_j & \text{if } u_j < 0. \end{cases}$$

Clearly $u = u^+ - u^-$ and $|u| = u^+ + u^-$. Since $\hat{c}^{\mathsf{T}}\hat{x} \geq \hat{c}^{\mathsf{T}}\tilde{x}$, we have $\hat{c}^{\mathsf{T}}u \geq 0$.

We prove this lemma by contradiction. Suppose that there exists $h \in [m] \setminus \mathcal{E}$ such that $\tilde{y}_h > nk$ and $a_h^{\mathsf{T}}\hat{x} \neq b_h$. Since $\hat{x} \in F$ and $a_h, \hat{x}, b_h$ are integral, we have $a_h^{\mathsf{T}}\hat{x} \leq b_h - 1$. We rewrite (10) as $A^{\mathsf{T}}\tilde{y} - \mathbf{1} \leq \hat{c} \leq A^{\mathsf{T}}\tilde{y} + \mathbf{1}$. Thus

$$\begin{aligned} \hat{c}^{\mathsf{T}}u = \hat{c}^{\mathsf{T}}u^+ - \hat{c}^{\mathsf{T}}u^- &\leq (A^{\mathsf{T}}\tilde{y} + \mathbf{1})^{\mathsf{T}}u^+ - (A^{\mathsf{T}}\tilde{y} - \mathbf{1})^{\mathsf{T}}u^- \\ &= (A^{\mathsf{T}}\tilde{y})^{\mathsf{T}}(u^+ - u^-) + \mathbf{1}^{\mathsf{T}}(u^+ + u^-) = (A^{\mathsf{T}}\tilde{y})^{\mathsf{T}}u + \mathbf{1}^{\mathsf{T}}|u|. \end{aligned} \tag{12}$$

We can upper bound $\mathbf{1}^{\mathsf{T}}|u|$ in (12) by observing that $|u_j| \leq k$ for all $j \in [n]$, since F is a lattice polytope in $[0, k]^n$ and u is the difference of two vectors in F. Thus

$$\mathbf{1}^{\mathsf{T}}|u| \leq nk. \tag{13}$$

We now compute an upper bound for $(A^\mathsf{T}\tilde{y})^\mathsf{T}u = \tilde{y}^\mathsf{T}Au$ in (12):

$$\begin{aligned}
\tilde{y}^\mathsf{T}Au &= \tilde{y}_h a_h^\mathsf{T}u + \sum_{i\in\mathcal{E}} \tilde{y}_i a_i^\mathsf{T}u + \sum_{\substack{i\in[m]\setminus\mathcal{E}\\ i\neq h}} \tilde{y}_i a_i^\mathsf{T}u \\
&< -nk + \sum_{i\in\mathcal{E}} \tilde{y}_i a_i^\mathsf{T}u + \sum_{\substack{i\in[m]\setminus\mathcal{E}\\ i\neq h}} \tilde{y}_i a_i^\mathsf{T}u && (14)\\
&= -nk + \sum_{\substack{i\in[m]\setminus\mathcal{E}\\ i\neq h}} \tilde{y}_i a_i^\mathsf{T}u && (15)\\
&\le -nk + \sum_{\substack{i\in[m]\setminus\mathcal{E}\\ i\neq h,\, \tilde{y}_i>0}} \tilde{y}_i a_i^\mathsf{T}u && (16)\\
&\le -nk. && (17)
\end{aligned}$$

To prove the strict inequality in (14) we show $\tilde{y}_h a_h^\mathsf{T}u < -nk$. We have $\tilde{y}_h > nk > 0$, thus condition (11) implies $a_h^\mathsf{T}\tilde{x} = b_h$. Since $a_h^\mathsf{T}\hat{x} \le b_h - 1$, we get $a_h^\mathsf{T}u = a_h^\mathsf{T}\hat{x} - a_h^\mathsf{T}\tilde{x} \le -1$. We multiply $\tilde{y}_h > nk$ by $a_h^\mathsf{T}u$ and obtain $\tilde{y}_h \cdot a_h^\mathsf{T}u < nk \cdot a_h^\mathsf{T}u \le -nk$. Equality (15) follows from the fact that, for each $i \in \mathcal{E}$, we have $a_i^\mathsf{T}\hat{x} = b_i$ and $a_i^\mathsf{T}\tilde{x} = b_i$ since both $\hat{x}$ and $\tilde{x}$ are in F, thus $a_i^\mathsf{T}u = 0$. Inequality (16) follows since $\tilde{y} \ge 0$. To see why inequality (17) holds, first note that, from condition (11), $\tilde{y}_i > 0$ implies $a_i^\mathsf{T}\tilde{x} = b_i$. Furthermore, since $\hat{x} \in F$, we have $a_i^\mathsf{T}\hat{x} \le b_i$. Hence we have $a_i^\mathsf{T}u \le 0$ and so $\tilde{y}_i a_i^\mathsf{T}u \le 0$.

By combining (12), (13) and (17) we obtain $\hat{c}^\mathsf{T}u < 0$. This is a contradiction since we have previously seen that $\hat{c}^\mathsf{T}u \ge 0$. □

Lemma 3.2 has a flavor similar to that of [28, Lem. 1.1]. However, there are three key differences. First, Lemma 3.2 deals with a primal in general form, while [28, Lem. 1.1] considers a primal in standard form. Second, Lemma 3.2 deals with a polytope in $[0, k]^n$ and exploits k as a parameter in condition (9), while [28, Lem. 1.1] exploits the maximum absolute value of a subdeterminant of the constraint matrix. It can be checked that these two results cannot be obtained from each other by switching from the general form representation to standard form, or vice versa. The third key difference is that the vector $\hat{x}$ in the statement of Lemma 3.2 is required to be integer, while this is *not* the case in [28, Lem. 1.1]. Essentially, we are able to detect if there is any constraint of $Ax \le b$ that is satisfied with equality by an integer vector in P that maximizes $c^\mathsf{T}x$. This result could be of independent interest in the field of integer programming.

For a vector $w \in \mathbb{R}^n$ and a polyhedron $Q \subseteq \mathbb{R}^n$, we say that a vector is *w-maximal in Q* if it maximizes $w^\mathsf{T}x$ over Q. We are now ready to show that, at each iteration, every optimal solution to (1) lies in F.

Lemma 3.3 *The set $\mathcal{E}$ updated in step 3 of the face-fixing algorithm is such that every vector x^* that is c-maximal in P satisfies $a_i^\mathsf{T}x^* = b_i$ for $i \in \mathcal{E}$.*

Proof It suffices to prove the statement for a vertex x^* of P 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 generic iteration. At the beginning of step 3 we have that x^* is c-maximal in F. Our goal is to prove that when we add an index $h \in \mathcal{H} \setminus \mathcal{E}$ to $\mathcal{E}$ at the end of step 3, we have that $a_h^\mathsf{T} x^* = b_h$.

First, note that x^* is also $\hat{c}$-maximal in F, as $\hat{c}$ is a scaling of $\bar{c}$. Since the vector $\tilde{x}$ computed in step 3 lies in F, we have $\hat{c}^\mathsf{T} x^* \geq \hat{c}^\mathsf{T} \tilde{x}$. Moreover, x^* is integral, since it is a vertex of P. Finally, for each $h \in \mathcal{H} \setminus \mathcal{E}$, we have $\tilde{y}_h > nk$. Thus Lemma 3.2 implies $a_h^\mathsf{T} x^* = b_h$ for all $h \in \mathcal{H} \setminus \mathcal{E}$. □

We are now ready to show that the face-fixing algorithm is correct.

Proposition 3.4 *The face-fixing algorithm returns an optimal solution to the LP problem* (1).

Proof Consider the face F defined when the algorithm terminates. Lemma 3.3 implies that F contains all optimal solutions to (1). Note that the affine hull of F is contained in $\{x \in \mathbb{R}^n \mid a_i^\mathsf{T} x = b_i \text{ for } i \in \mathcal{E}\}$. Hence, due to the termination condition, all vectors in F have the same objective value $c^\mathsf{T} x$. Since the vector x^* returned is in F, we obtain that x^* is an optimal solution to (1). □

3.2 Length of Simplex Path

To bound the length of the simplex path constructed by the face-fixing algorithm from the input vertex x^0 to an optimal vertex of (1) we exploit the fact that this path is obtained by merging together the different paths constructed by the scaling algorithm at each iteration.

Proof of Theorem 1.2 From Lemma 3.1 the face-fixing algorithm performs at most $n+1$ iterations. Each time the face-fixing algorithm performs step 3, it calls the scaling algorithm with input F, x^*, and $\tilde{c}$. Since F is a $[0, k]$-polytope, by Proposition 2.4, each time the scaling algorithm is called, it generates a simplex path of length at most $nk\big(\lceil \log \|\tilde{c}\|_\infty \rceil + 1\big)$, where $\|\tilde{c}\|_\infty = n^3 k\alpha$. Since $\log \|\tilde{c}\|_\infty \in O(\log(nk\alpha))$, each time we run the scaling algorithm, we generate a simplex path of length in $O(nk \log(nk\alpha))$. Therefore, the simplex path generated throughout the entire algorithm has length in $O(n^2 k \log(nk\alpha))$. □

We immediately obtain the following corollary of Theorem 1.2.

Corollary 3.5 *If α is polynomially bounded in n, k, then the length of the simplex path generated by the face-fixing algorithm is in $O(n^2 k \log(nk))$. If we also assume $k = 1$, the length reduces to $O(n^2 \log n)$.*

Next, we compare our bound on the length of the simplex path constructed by the face-fixing algorithm with other known bounds for some classic pivoting rules that can be applied to $[0, k]$-polytopes.

As discussed in Sect. 2.3, a result by Kitahara and Mizuno [17,18] implies that, by using classic pivoting rules, we can construct a simplex path in P whose length is at most $n^2 K \log(nK)$, where $K = \max\{k, S\}$ and $S = \max\{\|b - Ax\|_\infty \mid x \in P\}$. In particular, if each inequality of $Ax \le b$ is active at some vertex of P, we have that S and K are in $O(nk\alpha)$, thus in this case the upper bound implied by the result of Kitahara and Mizuno is in $O(n^3 k\alpha \log(nk\alpha))$. Note that the dependence on α is superlinear, while in our face-fixing algorithm it is only logarithmic. Our upper bound is better also for small values of α. In fact, for $\alpha = 1$, their upper bound is in $O(n^3 k \log(nk))$ and ours is in $O(n^2 k \log(nk))$.

To show that the bound of $O(nk\alpha)$ on S and K just discussed can be tight, we now provide an example of a [0, 1]-polytope with $\alpha = 1$ and $S \in \Omega(n)$. Consider the stable set polytope of a t-perfect graph $G = (V, E)$, that is defined by the vectors $x \in \mathbb{R}^V_+$ satisfying:

$$\begin{aligned} x_i + x_j &\le 1, && ij \in E, \\ \sum_{i \in V(C)} x_i &\le \left\lfloor \frac{|V(C)|}{2} \right\rfloor, && C \text{ odd cycle in } G, \end{aligned} \tag{18}$$

where $V(C)$ denotes the nodes in the odd cycle C [26]. Note that $x = 0$ is the characteristic vector of the empty stable set, thus it is a vertex of the stable set polytope. If G is an odd cycle on $|V| = n$ nodes, then G is t-perfect, and the inequality (18) corresponding to the cycle containing all nodes of G is facet-defining. Furthermore, the slack in such constraint can be as large as $\lfloor n/2 \rfloor$, therefore $S \in \Omega(n)$. Consequently, the upper bound implied by [17,18] is in $\Omega(n^3 \log n)$, while the upper bound given by our face-fixing algorithm is in $O(n^2 \log n)$.

In a subsequent paper [22], Mizuno proposed an algorithm that can be used to trace a simplex path on P whose length is at most $O(n^3 m^4 \Delta^3 \log(n^2 m^3 \Delta^3))$, where Δ is the largest absolute value of a subdeterminant of the matrix A. Note that, while the upper bound on the length of the simplex path generated by the face-fixing algorithm depends on parameters n, k, α, the upper bound on the length of the simplex path generated by this algorithm depends on parameters n, m, Δ. Therefore these two bounds are not directly comparable. In particular, it is known that Δ can grow as $\alpha^n \cdot n^{n/2}$. Moreover, as remarked earlier, even in [0, 1]-polytopes m can grow exponentially in n (see, e.g., [26]).

3.3 Complexity

In this last section, we bound the number of operations performed by the face-fixing algorithm to construct the next vertex in the simplex path or to certify optimality of the current vertex. First, we upper bound the number of operations needed to compute an optimal solution $\tilde{y}$ to the dual ($\tilde{\text{D}}$) with the properties stated in step 3.

Lemma 3.6 *In step* 3 *of the face-fixing algorithm, a vector* $\tilde{y}$ *satisfying* (i) *and* (ii) *can be computed in a number of operations that is polynomially bounded in* size A. *If* P *is simple, the number of operations is in* $O(nm + n^3)$.

Proof First, assume that the polytope P is simple. Let problem ($\tilde{\text{P}}$)$'$ be obtained from ($\tilde{\text{P}}$) by dropping the inequalities that are not active at $\tilde{x}$. This can be done in

$O(nm)$ operations. Since P is simple, the number of constraints in $(\tilde{\mathrm{P}})'$ is n and the $n \times n$ constraint matrix $\tilde{A}$ of $(\tilde{\mathrm{P}})'$ is invertible. Note that $\tilde{x}$ is an optimal solution to $(\tilde{\mathrm{P}})'$ as well. Let $(\tilde{\mathrm{D}})'$ be the dual of $(\tilde{\mathrm{P}})'$. Note that $(\tilde{\mathrm{D}})'$ is obtained from $(\tilde{\mathrm{D}})$ by dropping the variables y_j corresponding to the inequalities of $(\tilde{\mathrm{P}})$ dropped to obtain $(\tilde{\mathrm{P}})'$. Since $(\tilde{\mathrm{P}})'$ has an optimal solution, then so does $(\tilde{\mathrm{D}})'$ from strong duality. The constraint matrix of $(\tilde{\mathrm{D}})'$ is the invertible matrix $\tilde{A}^\mathsf{T}$. The only feasible solution to the system of linear equations in $(\tilde{\mathrm{D}})'$ is the vector $\tilde{y}' := \tilde{A}^{-\mathsf{T}}\tilde{c}$ which can be computed in $O(n^3)$ operations. Since $(\tilde{\mathrm{D}})'$ is feasible, then $\tilde{y}'$ must satisfy all the constraints in $(\tilde{\mathrm{D}})'$, thus $\tilde{y}'$ is optimal. 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 $(\tilde{\mathrm{D}})$, and, from complementary slackness with $\tilde{x}$, it is optimal to $(\tilde{\mathrm{D}})$. Furthermore, $\tilde{y}$ clearly satisfies (i). To see that it satisfies (ii), note that the equalities $a_i^\mathsf{T} x = b_i$, $i \in \mathcal{E}$, are all in $(\tilde{\mathrm{P}})'$. Since the constraints in $(\tilde{\mathrm{P}})'$ are all linearly independent, problem $(\tilde{\mathrm{P}})'$ cannot contain any constraint $a_j^\mathsf{T} x \le b_j$, for $j \in [m] \setminus \mathcal{E}$ such that a_j can be written as a linear combination of a_i, $i \in \mathcal{E}$. Hence, the corresponding dual variable $\tilde{y}_j$ has been set to zero.

Consider now the general case where P need not be simple. First, we show how to compute a vector $\tilde{y}$ that satisfies (i). Since $(\tilde{\mathrm{P}})$ has an optimal solution, then so does $(\tilde{\mathrm{D}})$ from strong duality. Let $(\tilde{\mathrm{D}})'$ be obtained from $(\tilde{\mathrm{D}})$ 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 $(\tilde{\mathrm{D}})$ and $(\tilde{\mathrm{D}})'$ are equivalent, so $(\tilde{\mathrm{D}})'$ has an optimal solution. Furthermore, since $(\tilde{\mathrm{D}})'$ 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 a number of operations polynomially bounded in size A. Let $\tilde{y}$ be obtained from $\tilde{y}'$ by replacing each pair $\tilde{y}_i'^+$, $\tilde{y}_i'^-$ with $\tilde{y}_i := \tilde{y}_i'^+ - \tilde{y}_i'^-$. It is simple to check that $\tilde{y}$ is an optimal solution to $(\tilde{\mathrm{D}})$. 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 $(\tilde{\mathrm{P}})'$ be obtained from $(\tilde{\mathrm{P}})$ by dropping the inequalities $a_j^\mathsf{T} x \le b_j$, 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 $(\tilde{\mathrm{P}})$ is feasible, then $(\tilde{\mathrm{P}})$ and $(\tilde{\mathrm{P}})'$ have the same feasible region and are therefore equivalent. Let $(\tilde{\mathrm{D}})'$ be the dual of $(\tilde{\mathrm{P}})'$. Note that $(\tilde{\mathrm{D}})'$ is obtained from $(\tilde{\mathrm{D}})$ by dropping the variables y_j corresponding to the inequalities of $(\tilde{\mathrm{P}})$ dropped to obtain $(\tilde{\mathrm{P}})'$. Note that $(\tilde{\mathrm{P}})'$ has the same form as that of $(\tilde{\mathrm{P}})$, thus, from the the first part of the proof, we can compute a vector $\tilde{y}'$ optimal to $(\tilde{\mathrm{D}})'$ with at most n nonzero components. Furthermore, $\tilde{y}'$ can be computed in a number of operations polynomially bounded in size 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 $(\tilde{\mathrm{D}})$, and, from complementary slackness with $\tilde{x}$, it is optimal to $(\tilde{\mathrm{D}})$. Furthermore, $\tilde{y}$ satisfies (i) and (ii). □

Proposition 3.7 *The number of operations performed by the face-fixing algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in* size A *and* $\log k$. *If P is simple, the number of operations is in $O(n^4 + n^2 m \log^2(nk\alpha))$.*

Proof First, we discuss the number of operations performed in a single iteration of the face-fixing algorithm:

(a) In step 1, computing the projection $\bar{c}$ of c onto the subspace $\{x \in \mathbb{R}^n \mid a_i^\top x = 0 \text{ for } i \in \mathcal{E}\}$ can be done in $O(n^3)$ operations via Gaussian elimination.
(b) 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 $n \log \|\tilde{c}\|_\infty$.
(c) In step 3 we call the scaling algorithm to compute the vector $\tilde{x}$. From Lemma 2.6, the number of operations performed to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in size A and $\log \|\tilde{c}\|_\infty$. If P is simple, the number of operations is $O(nm \log^2 \|\tilde{c}\|_\infty)$.
(d) At the end of step 3 we compute the vector $\tilde{y}$. From Lemma 3.6, the number of operations performed to compute this vector is polynomially bounded in size A, and by $O(nm + n^3)$ if P is simple.

Recall from Lemma 3.1 that the face-fixing algorithm performs at most $n+1$ iterations. Moreover, each vector $\tilde{c}$ computed at step 2 is such that $\log \|\tilde{c}\|_\infty \in O(\log (nk\alpha))$.

To construct the next vertex in the simplex path or to certify optimality of the current vertex, in the worst case the face-fixing algorithm calls n times the scaling algorithm. Therefore, the number of operations is bounded by the product of n with the sum of the operations bounds in (a)–(d) above. In the general case, this number is polynomially bounded in size A and $\log \|\tilde{c}\|_\infty$. If P is simple, this number is bounded by

$$O\big(n \cdot (n^3 + n \log \|\tilde{c}\|_\infty + nm \log^2 \|\tilde{c}\|_\infty + nm + n^3)\big) \in O(n^4 + n^2 m \log^2 \|\tilde{c}\|_\infty).$$

The statement follows since size A is polynomial in $n, m, \log \alpha$, and $\log \|\tilde{c}\|_\infty \in O(\log (nk\alpha))$. □

The following proposition shows that if the polytope P is "well described", then the number of operations performed by both the preprocessing & scaling algorithm and the face-fixing algorithm to construct the next vertex in the simplex path is polynomially bounded in $n, m, \log k$. In particular, it is independent from α. Formally, we say that a full-dimensional polytope $P = \{x \in \mathbb{R}^n \mid Ax \le b\}$ is *well described* by the system $Ax \le b$ if each inequality in $Ax \le b$ is facet-defining, and the greatest common divisor of the entries in each row of A is one. It is well known that given a system $A'x \le b'$ describing a full-dimensional polytope P, we can obtain in polynomial time a system $Ax \le b$ such that P is well described by $Ax \le b$.

Proposition 3.8 *Assume that P is well described. Then, the number of operations performed by the preprocessing & scaling algorithm and by the face-fixing algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in $n, m, \log k$.*

Proof From Propositions 2.7 and 3.7, the number of operations performed by either algorithm to construct the next vertex in the simplex path, or to certify optimality of the current vertex, is polynomially bounded in size A and $\log k$. Recall that size A is polynomial in n, m, and $\log \alpha$. Therefore, it suffices to show that $\log \alpha$ is polynomially bounded in n and $\log k$.

Denote by φ the facet complexity of P and by ν the vertex complexity of P. From [25, Thm. 10.2], we know that φ and ν are polynomially related, and in particular

$\varphi \le 4n^2\nu$. Since P is a $[0,k]$-polytope, we have $\nu \le n \log k$. Due to the assumptions in the statement of the proposition, and [7, Rem. 1.1], we obtain that $\log \alpha \le n\varphi$. Hence, $\log \alpha \le n\varphi \le 4n^3\nu \le 4n^4 \log k$. □

We highlight that all the obtained bounds on the number of operations performed by our algorithms to construct the next vertex in the simplex path also depend on the number m of inequalities in the system $Ax \le b$. This is in contrast with the lengths of the simplex paths, which only depend on n and k. This is because, in order to determine the next vertex, the algorithm needs to read all the inequalities defining the polytope, thus the number of operations must depend also on m.

The total number of operations performed by our algorithms can be simply obtained by multiplying the length of the simplex path with the number of operations performed to construct the next vertex in the simplex path. If we assume that P is well described and that k is polynomially bounded in n and m, Proposition 3.8, Theorems 1.1 and 1.2 imply that the preprocessing & scaling algorithm and the face-fixing algorithm run in strongly polynomial time.

To conclude, we remark that our algorithms can be used to solve an LP problem whose feasible set is a lattice polytope, provided that a starting vertex is given in input. It should be noted that in the face-fixing algorithm we use Tardos' algorithm to compute an optimal dual solution in step 3, if the lattice polytope is not simple. Thus, the number of operations performed by our algorithm is clearly larger than that of Tardos' algorithm. This is to be expected, since our overall goal is not only of solving the given LP problem, but also of tracing a simplex path from the starting vertex to an optimal vertex.

Data Availability Statement Data sharing not applicable to this article as no datasets were generated or analyzed during the current study.

References

1. Acketa, D.M., Žunić, J.D.: On the maximal number of edges of convex digital polygons included into an $m \times m$-grid. J. Combin. Theory Ser. A **69**(2), 358–368 (1995)
2. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows: Theory, Algorithms, and Applications. Prentice Hall, Englewood Cliffs (1993)
3. Alon, N., Vu, V.H.: Anti-Hadamard matrices, coin weighing, threshold gates, and indecomposable hypergraphs. J. Combin. Theory Ser. A **79**(1), 133–160 (1997)
4. Balog, A., Bárány, I.: On the convex hull of the integer points in a disc. In: 7th Annual Symposium on Computational Geometry (North Conway 1991), pp. 162–165. ACM, New York (1991)
5. Bertsimas, D., Tsitsiklis, J.N.: Introduction to Linear Optimization. Athena Scientific, Belmont (1997)
6. Blanchard, M., De Loera, J., Louveaux, Q.: On the length of monotone paths in polyhedra (2020). arXiv:2001.09575
7. Conforti, M., Cornuéjols, G., Zambelli, G.: Integer Programming. Graduate Texts in Mathematics, vol. 271. Springer, Cham (2014)
8. Del Pia, A., Michini, C.: On the diameter of lattice polytopes. Discrete Comput. Geom. **55**(3), 681–687 (2016)
9. Deza, A., Manoussakis, G., Onn, S.: Primitive zonotopes. Discrete Comput. Geom. **60**(1), 27–39 (2018)
10. Deza, A., Pournin, L.: Improved bounds on the diameter of lattice polytopes. Acta Math. Hungar. **154**(2), 457–469 (2018)
11. Deza, A., Pournin, L.: Primitive point packing (2020). arXiv:2006.14228

12. Deza, A., Pournin, L., Sukegawa, N.: The diameter of lattice zonotopes. Proc. Am. Math. Soc. **148**(8), 3507–3516 (2020)
13. Dinic, E.A.: An algorithm for the step-by-step decrease of discrepancies, and transport problems. In: Issled. Diskret. Mat., pp. 46–57. Nauka, Moscow (1973). (in Russian)
14. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. J. Assoc. Comput. Mach. **19**(2), 248–264 (1972)
15. Frank, A., Tardos, É.: An application of simultaneous Diophantine approximation in combinatorial optimization. Combinatorica **7**(1), 49–65 (1987)
16. Gabow, H.N.: Scaling algorithms for network problems. J. Comput. Syst. Sci. **31**(2), 148–168 (1985)
17. Kitahara, T., Mizuno, S.: On the number of solutions generated by the dual simplex method. Oper. Res. Lett. **40**(3), 172–174 (2012)
18. Kitahara, T., Mizuno, S.: A bound for the number of different basic solutions generated by the simplex method. Math. Program. Ser. A **137**(1–2), 579–586 (2013)
19. Kleinschmidt, P., Onn, S.: On the diameter of convex polytopes. Discrete Math. **102**(1), 75–77 (1992)
20. Le Bodic, P., Pavelka, J.W., Pfetsch, M.E., Pokutta, S.: Solving MIPs via scaling-based augmentation. Discrete Optim. **27**, 1–25 (2018)
21. Lenstra, A.K., Lenstra Jr., H.W., Lovász, L.: Factoring polynomials with rational coefficients. Math. Ann. **261**(4), 515–534 (1982)
22. Mizuno, S.: The simplex method using Tardos' basic algorithm is strongly polynomial for totally unimodular LP under nondegeneracy assumption. Optim. Methods Softw. **31**(6), 1298–1304 (2016)
23. Naddef, D.: The Hirsch conjecture is true for (0, 1)-polytopes. Math. Program. **45**(1), 109–110 (1989)
24. Pokutta, S.: Restarting algorithms: sometimes there is free lunch. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (Vienna 2020), pp. 22–38. Springer, Cham (2020)
25. Schrijver, A.: Theory of Linear and Integer Programming. Wiley-Interscience Series in Discrete Mathematics. Wiley, Chichester (1986)
26. Schrijver, A.: Combinatorial Optimization. Polyhedra and Efficiency. Springer, Berlin (2003)
27. Schulz, A.S., Weismantel, R., Ziegler, G.M.: 0/1-Integer programming: optimization and augmentation are equivalent. In: Algorithms—ESA '95 (Corfu 1995). Lecture Notes in Comput. Sci., vol. 979, pp. 473–483. Springer, Berlin (1995)
28. Tardos, É.: A strongly polynomial algorithm to solve combinatorial linear programs. Oper. Res. **34**(2), 250–256 (1986)
29. Thiele, T.: Extremalprobleme für Punktmengen. PhD thesis, Freie Universität Berlin (1991)
30. Ziegler, G.M.: Lectures on 0/1-polytopes. In: Polytopes—Combinatorics and Computation (Oberwolfach 1997). DMV Sem., vol. 29, pp. 1–41. Birkhäuser, Basel (2000)