A polynomial algorithm for linear optimization which is strongly polynomial under certain conditions on optimal solutions

Sergei Chubanov*

University of Siegen, Germany

e-mail: sergei.chubanov@uni-siegen.de

January 31, 2015

Abstract

This paper proposes a polynomial algorithm for linear programming which is strongly polynomial for linear optimization problems $\min\{c^T x : Ax = b, x \geq \mathbf{0}\}$ having optimal solutions where each non-zero component x_j belongs to an interval of the form $[\alpha_j, \alpha_j \cdot 2^{p(n)}]$, where α_j is some positive value and $p(n)$ is a polynomial of the number of variables. We do not make any additional assumptions about c and A. This class of problems includes linear optimization problems having 0-1 optimal solutions and the Markov Decision Problem with a fixed discount factor as special cases.

Keywords: Linear programming, polynomial algorithm, strongly polynomial algorithm.

*Chubanov, S.; Universität Siegen, Fakultät III, Kohlbettstr. 15, 57068 Siegen, Deutschland.

1 Introduction

In this paper we present a new polynomial algorithm for linear programming. This algorithm is strongly polynomial for linear optimization problems (in the standard form) having optimal solutions x^* where each component x^*_j belongs to $\{0\} \cup [\alpha_j, \alpha_j \cdot 2^{p(n)}]$. Here, n is the number of variables, α_j are positive real values known in advance, and $p(n)$ is a polynomial of n. This class includes linear optimization problems having 0-1 optimal solutions (e.g., the assignment problem and the min-cut problem can be represented in this form) and the Markov Decision Problem with a fixed discount factor (MDP). Ye [27] has shown that the MDP can be solved in strongly polynomial time by a modification of the barrier method exploiting a special structure of the coefficient matrix of the MDP. The class of problems proved to be solvable in strongly polynomial time in the present paper is much more general because we do not make any additional assumptions on the coefficient matrix.

In the general case it is an open question whether there exists a strongly polynomial algorithm for linear programming, i.e., a polynomial algorithm whose running time (the number of arithmetic operations, comparisons, and variable assignments) would be bounded by a polynomial depending exclusively on the number of variables and constraints. In a slightly different form, this open problem is known as Smale's 9th problem [23], which is formulated in terms of the real model of computation where a real number is considered to be of the unit size. The real model of computation is often referred to as the BSS model of computation (BSS stands for Blum, Shub, and Smale [3]). Smale's 9th problem asks for a polynomial algorithm under the real model of computation. Here we should note that an algorithm which is polynomial in the real model of computation does not immediately imply an algorithm which would be strongly polynomial in the usual sense because of the additional requirement that the space used by the algorithm is polynomially bounded when the problem is posed over rational numbers.

Actually, the complexity result of Ye [27] is formulated in terms of the real model of computation and it is not clear if Ye's algorithm can be transformed into an algorithm that would be strongly polynomial in the usual sense. On the other hand, Ye [28] gave a variant

of the simplex method which is a strongly polynomial algorithm for the MDP. However, this variant of the simplex method is not polynomial for a more general case where the so-called discount factor is assumed to be of strongly polynomial size, while Ye's variant of the barrier method runs in strongly polynomial time even under this more general assumption. In this case, our algorithm is strongly polynomial in the usual sense.

One of the major results in the area of strongly polynomial algorithms belongs to Tardos [24]. She proposed a method of converting any polynomial algorithm for linear programming to a strongly polynomial algorithm for linear optimization problems where the entries of the coefficient matrices are integers with the binary sizes bounded by a polynomial in the dimension. That is, the algorithm proposed in the present paper can also be converted to a strongly polynomial algorithm for such problems.

Vavasis and Ye [25] developed a variant of the barrier method whose running time is bounded by a polynomial depending only on the coefficient matrix. As a corollary, they obtain the above result of Tardos.

Another result on strongly polynomial solvability in linear programming belongs to Megiddo [17]. He considered systems of linear inequalities with coefficient matrices having at most two nonzero entries in each row. This is an example of a class of problems which is neither a subset of the class of linear problems proved to be solvable in strongly polynomial time in the present paper nor a subset of the class considered by Tardos.

The paper is organized as follows. In Section 2 we formulate the problem and define some classes of linear problems that are solvable in strongly polynomial time by our algorithm. This section also discusses the decision version of the problem and some related results such as a polynomial algorithm for linear feasibility problems proposed by Chubanov [7] and the algorithm of Végh and Zambelli [26] based on an earlier algorithm of Chubanov [6]. In Section 3 we present a polynomial algorithm for linear programming which is strongly polynomial under the conditions formulated at the beginning of the paper. Moreover, we discuss our algorithm in the context of finding lower bounds for some nonlinear problems.

The algorithms mentioned above (except for that being the subject of this paper) and all traditional polynomial algorithms for linear programming, like the ellipsoid method proved

to be polynomial by Khachiyan [14] or interior-point methods (see Karmarkar [13], Gill et al. [11], and Renegar [20]), generate a sequence of points in the vector space of the original problem. In contrast to these methods, the algorithm presented in this paper generates a sequence in the vector space of matrices of the respective dimension. In section 2.5 we give a general algorithm of this type. This algorithm is also applicable to more general convex problems.

2 Formulation of the problem and preliminaries

Let $\mathbf{1}$ denote an all-one vector and $\mathbf{0}$ denote a zero vector. Let $\mathbf{e}_j$ denote the unit vector whose jth component is equal to 1 (the other components are zero). The dimensions of $\mathbf{1}$, $\mathbf{0}$, and $\mathbf{e}_j$ will be determined by the context.

For two matrices X_1 and X_2, of the same dimension, $\max(X_1, X_2)$ will denote their componentwise maximum.

We will write $S + v$, for a set S and a vector v in a Euclidean space, to denote the Minkowski sum $S + \{v\} = \{w + v : w \in S\}$.

2.1 Formulation of the problem

Let a linear optimization problem be written in the standard form:

$$\min\{c^T x : Ax = b, x \geq \mathbf{0}\}.$$

Here, A is a real $m \times n$ matrix of full rank, b is a real m-vector, and c is a real n-vector.

Our algorithm for this problem is polynomial in the usual sense when A, b, and c are rational. Moreover, its time complexity is strongly polynomial if the problem is either infeasible or has the following property:

(I) There exists an optimal solution x^* where $x^*_j \in \{0\} \cup [\alpha_j, \alpha_j \cdot 2^{p(n)}]$ for all j.

Here, α_j are nonnegative values and p is a polynomial. We assume that both the values α_j and the polynomial p are known in advance. Note that we do not impose any additional

constraints on c and A. Further, we refer to this problem as to the problem with (I) and to the respective solutions x^* as to solutions with (I).

The above condition (I) is partially motivated by the following special cases:

(i) Linear optimization problems which are either infeasible or have 0-1 optimal solutions. (Well-known examples of problems that can be represented in this form and can be solved in strongly polynomial time include the assignment problem and the minimum-cut problem.)

(ii) The Markov Decision Problem with a fixed discount factor (MDP). This problem arises in the theory of Markov decision processes with discrete time and is formulated as a linear optimization problem in the standard form with a special structure of the coefficient matrix. Ye [27] has proved that this problem is solvable in strongly polynomial time.

For a problem having 0-1 optimal solutions we can set $\alpha_j := 1$ and $p(n) := 1$. The MDP has an optimal solution x^* such that $x^*_j \leq \frac{n}{1-\theta}$ and either $x^*_j = 0$ or $x^*_j \geq 1$ for all j; see Lemma 3.2 in [27]. Here, $\theta \in [0, 1)$ is a constant called the discount factor; see Ye [27] and Dantzig [8] for details. To see that the MDP is a subset of our problem, we let $\alpha_j := 1$ and $p(n) := \log \frac{n}{1-\theta}$.

Ye [27] was first to develop a strongly polynomial algorithm for the MDP. Ye's algorithm for the MDP is based on the barrier method and uses the special structure of the coefficient matrix of the MDP. Ye's analysis of the barrier method is quite complicated and one can hardly hope for a simpler strongly polynomial barrier method for the more general problem with (I); so to the best of our knowledge it is not known if the barrier method can be modified to solve the problem with (I) in strongly polynomial time.

2.2 Ellipsoid method with a projection step for a case with binary solutions

Now we will show that the ellipsoid method can be modified to run in strongly polynomial time for the decision version of the problem (i.e, with $c = \mathbf{0}$) in the special case (i). At the

same time, this modification is to illustrate that it can be difficult, if at all possible, to find a reasonable modification of the ellipsoid method to solve linear optimization problems with (I) in strongly polynomial time.

The decision version of our linear optimization problem is simply the inequality system

$$Ax = b, x \geq \mathbf{0}.$$

In this case the condition (i) means that the system is either infeasible or has a 0-1 solution. First of all, we restrict the class of ellipsoids that we are allowed to consider to those of the form

$$E(M, z) = \{x : (x - z)^T M^{-2}(x - z) \leq 1\},$$

where M is a positive definite diagonal matrix and z is the center of the ellipsoid. Assume that the current ellipsoid contains a feasible 0-1 solution if the problem is feasible. Our modification of the ellipsoid method works as follows:

1. If $\det M < 1/2^n$, then the ellipsoid has no intersection with at least one of the facets of the unit cube $[0, 1]^n$. To find one of such facets, we consider $M_{jj} = \min_i M_{ii}$. Since $\det M < 1/2^n$, it follows that $M_{jj} < 1/2$. Then $|x_j - z_j| \leq M_{jj} \leq 1/2$ for all 0-1 feasible solutions x because they belong to the current ellipsoid. If $z_j < 1/2$, then $x_j = 0$ for all 0-1 solutions x. Otherwise, $x_j = 1$ for all 0-1 solutions. Let $x_j := 0$ or $x_j := 1$, depending on the value of z_j. That is, we have fixed the variable x_j and reduced the dimension of the problem. The obtained problem must have 0-1 solutions if the original problem has 0-1 solutions. Now we can consider an ellipsoid containing 0-1 solutions in $\mathbb{R}^{n-1}$. (If necessary, the obtained system should be transformed so as to preserve the property that the coefficient matrix is of full rank.)

2. If $Az = b$ and $z \geq \mathbf{0}$, then return z as a feasible solution. If $Az = b$ and a non-negativity constraint is violated at the current center z, i.e., if $z_j < 0$ for some j, then perform a usual iteration of the ellipsoid method with respect to the violated constraint. This step changes the matrix M and computes a new center z. Note that we obtain an ellipsoid of the same class as above, i.e., the new matrix M is again a positive definite diagonal matrix.

3. Otherwise, project z onto the affine subspace $\{x : Ax = b\}$ in the metric induced by the respective norm defined as $\|x\|_{M^{-1}} = \sqrt{x^T M^{-2} x}$ for any vector x. This means that the center of the next ellipsoid is computed as

$$z' := z - M^2 A^T (AM^2 A^T)^{-1} Az + M^2 A^T (AM^2 A^T)^{-1} b.$$

The matrix M remains the same at this step. This step translates the old ellipsoid with the center z to the new ellipsoid with the center z'. Also, this step preserves the property that the ellipsoid contains a feasible 0-1 solution because

$$E(M, z) \cap \{x : Ax = b\} \subseteq E(M, z'),$$

which is verified by a direct calculation.

The above modified ellipsoid method requires at most $O(n)$ consecutive repetitions of steps 2 and 3 in order to reduce the determinant of M by at least a factor of 2, which follows from the traditional ellipsoid method. Let the algorithm start with $z = \mathbf{0}$ and $M = \sqrt{n}I$, where I is the identity matrix. The choice of M ensures that the initial ellipsoid contains all 0-1 solutions. Then the algorithm performs no more than $O(n^2 \log n)$ consecutive iterations of steps 2 and 3 until $\det M < 1/2^n$, because, at the initial step, $\det M = \sqrt{n}^n$. Then step 1 fixes a variable and we perform the same procedure in $\mathbb{R}^{n-1}$. So we require $O(n^2 \log n)$ time per variable. An iteration can be implemented to run in $O(n^3)$ time. We obtain a running time of $O(n^6 \log n)$ to either find a solution or prove that there are no 0-1 solutions. We skip a work on the sizes of the numbers because this would take us away from the main goal of this paper.

Now, let us try to find out whether it is possible to modify the above method in order to obtain a strongly polynomial running time for the decision version of the problem with (I). Again, c is assumed to be a zero vector and our objective is to find a feasible solution. Assume without loss of generality that there exist feasible solutions in the unit cube $[0, 1]^n$. (If necessary, we can rescale the system because we know the polynomial $p(n)$ and the values α_j in (I).) In a strongly polynomial time we come to an ellipsoid whose volume is smaller than $1/2^{p(n)+1}$. Then we choose $M_{jj} = \min_i M_{ii}$. If $z_j \leq 1/2^{p(n)+1}$, we can conclude that

$x_j = 0$ for all feasible solutions. Otherwise, if $z_j > 1/2^{p(n)+1}$, it is hard to say something about the structure of solutions. This suggests that we need to impose some constraints on the class of ellipsoids, in addition to those we already have. For instance, it would be sufficient for such an approach to work if we periodically had $z_j \leq 1/2^{p(n)+1}$ and the number of iterations between two consecutive iterations with this property was strongly polynomial.

The above modified ellipsoid method can be viewed as a "parallelepiped" method constructing parallelepipeds of the form $\{x : |x_j - z_j| \leq M_{jj}, j = 1, \ldots, n\}$, if we consider circumscribing parallelepipeds in place of ellipsoids. It should be noted that in 1988 Nemirovsky [19] developed a polynomial algorithm for linear optimization, on the basis of a gradient method for convex quadratic optimization, which also constructs a sequence of parallelepipeds, of the above form, containing a solution of the problem. The volumes of the parallelepipeds in the sequence decrease in a geometric progression, which implies a polynomial running time. (I thank Kees Roos for pointing out that paper to me.) This algorithm is not suitable for solving the problem with (I) in strongly polynomial time, either, by the same reason as in the case of the modified ellipsoid method. The reason is clear. Even if a problem has the property (I), the number of parallelepipeds needed to determine a solution is not necessarily strongly polynomial.

2.3 Feasibility problems with (I)

At the same time, the decision version of the problem with (I) can be solved in strongly polynomial time by the algorithm of Chubanov [7] (note that Roos [21] recently proposed an improvement of this algorithm) or by the algorithm of Végh and Zambelli [26] mentioned in Section 1. The following theorem is a simple corollary of [7]:

Theorem 2.1 *A linear system $Ax = b, x \geq \mathbf{0}$, having solutions x^* such that $x^*_j \in \{0\} \cup [\alpha_j, \alpha_j 2^{p(n)}]$ can be solved in time $O(n^4 \cdot p(n))$.*

Proof. First of all, scale the system by replacing A by A $\mathrm{diag}(\alpha_1 2^{p(n)}, \ldots, \alpha_n 2^{p(n)})$. The obtained system has solutions in the unit cube $[0, 1]^n$. Apply the algorithm presented in [7]. At each iteration this algorithm calls a procedure, which is called the basic procedure in [7],

that either finds a feasible solution or proves that $x_j \leq 1/2$ for all feasible solutions x in $[0,1]^n$. If a solution is found, we convert it to a solution of the original problem. Otherwise, we divide the jth column of the current coefficient matrix by 2. If x' in $[0,1]^n$ is a solution of the previous system, then x'' with $x''_j = 2x'_j$, and the other components equal to the respective components of x, is feasible for the obtained system. The solution x'' belongs to $[0,1]^n$ because $x'_j \leq 1/2$, as proved by the basic procedure. Repeat the above step with respect to the current system. If the number of divisions of the same column of the coefficient matrix exceeds $p(n)$, we conclude that the current system has a solution with $x_j = 0$. Then we add the respective constraint to the system. As shown in [7], the overall running time of the basic procedure in the course of this process is $O(Kn^3)$, where K is the number of calls to the basic procedure. The running time of $O(n^4 \cdot p(n))$ follows from $K \leq np(n)$. ∎

In the above case, the algorithm of Végh and Zambelli [26] runs in time $O(n^5 p(n)/\log n)$. Given a parallelepiped of the form $\{x : \mathbf{0} \leq x \leq w\}$ containing a feasible solution, their algorithm determines a parallelepiped contained in a half of the previous one in time $O(n^4)$. This implies a running time of $O(n^5 p(n))$. This running time can be improved to $O(n^5 p(n)/\log n)$; see [26]. Note that the scaling process in the proof of the above theorem can also be interpreted as a procedure constructing a sequence of parallelepipeds of the above form.

Basu, Junod, and De Loera [2] propose a detailed study of the procedure lying in the basis of the algorithm in [6] and develop an algorithm for systems of linear inequalities of the form $Ax = b, \mathbf{0} \leq x \leq \lambda\mathbf{1}$, where A is totally unimodular and b is an integer vector. This problem is a subset of the problems considered by Tardos [24] mentioned in Section 1. The running time of the algorithm of Basu, Junod, and De Loera is polynomial in the dimension and in λ. They give an estimate of the running time of at least $O(\lambda^2 n^5)$, where n is the number of variables.

The above problem $Ax = b, \mathbf{0} \leq x \leq \lambda\mathbf{1}$, where A is totally unimodular, can be solved by the polynomial-time algorithm for linear problems with totally modular coefficient matrices proposed by Maurras, Truemper, and Akgül [16] in time $O(d_2(\lambda\mathbf{1})n^{10}\log n)$, where $d_2(\cdot)$ denotes the number of ones in the binary representation of λ, provided that λ is an integer. Note that if the system is feasible, then $\|b\|_\infty \leq \lambda n$. So if this condition is not satisfied, we

conclude that the system is infeasible. Otherwise, since b is an integer vector, the binary size of b is bounded by $n(\text{size}(\lambda) + \text{size}(n))$, where size denotes the binary size. That is, any polynomial algorithm for linear programming can solve the above feasibility problem in time bounded by a polynomial in n and in size(λ).

We can write the above system as $Ax = b, x + y = \lambda\mathbf{1}, x \geq \mathbf{0}, y \geq \mathbf{0}$. The coefficient matrix of $Ax = b, x + y = \lambda\mathbf{1}$, is totally unimodular. Using Cramer's rule, we can prove that if the system is feasible then there exists a solution where each component is an integer linear combination of λ and the components of b. Since b is an integer vector, each component of such a solution has the form $k + \lambda r$, where k and r are integers. Let λ be represented in the form $\lambda = r_1/r_2$ where r_1 and r_2 are integers. It follows that if the above system is feasible, then there exist solutions with x_j and y_j in $\{0\} \cup [1/r_2, \lambda]$, for all $j \in \{1, \ldots, n\}$. Then the algorithm used in the proof of Theorem 2.1 can solve the system in time $O(n^4 \text{size}(r_1))$.

However, it is not clear if the above approaches can work in the general case of the optimization problem with (I), where c can be any vector and A can be any matrix. An obvious way to include the objective function is to consider a sequence of systems $Ax = b, c^T x = \xi, x \geq \mathbf{0}$, with different values of ξ, but it is not clear how to choose the values ξ so that the number of systems would by strongly polynomial.

2.4 Scaling scheme

In a very general form, our algorithm follows a scaling scheme that is similar to that used in Chubanov [7]. It consists of scaling the coefficient matrix and the objective function whenever we find a suitable upper bound on a variable. The main ingredient of our algorithm is a polynomial procedure which is able to find such bounds.

Assume without loss of generality that the optimal solutions satisfying (I) lie in the unit cube $[0, 1]^n$. Let J be initialized with $\emptyset$. Our algorithm performs the following steps:

- In strongly polynomial time, find an optimal solution or an index $j \in \{1, \ldots, n\} \setminus J$ such that $x^*_j \leq 1/2$ for all optimal solutions x^* with (I). This is a key step which we discuss in more detail in Section 2.5.

- Divide the jth column of A by 2. The new coefficient matrix is AM, where M is obtained from the identity matrix by dividing the jth column by 2. If x' is feasible for the original problem, then $x'' = M^{-1}x'$ is a feasible solution of the problem

$$\min\{c^T Mx : AMx = b, x \geq \mathbf{0}\}.$$

  The optimal value of this problem is equal to the optimal value of the original problem. Since $x''_j = 2x'_j$, it follows that $x''_j \leq 1$ if x' is an optimal solution with (I). Thus, the new problem has an optimal solution with (I).

  In other words, we have applied an operation that is further called scaling. It consists of the replacement of the original linear optimization problem by the new one where both the coefficient matrix and the cost vector are multiplied by a positive definite diagonal matrix.

- Repeat the above steps with respect to the current problem. Take into account that if the number of divisions of the same column exceeds $p(n)$, then the respective variable is zero in the optimal solutions with (I). In this case add the respective equation to the formulation of the problem and add the respective index to J. In some number of steps we come to an optimal solution.

It is obvious that if we have a strongly polynomial procedure to perform the first step, then we obtain a strongly polynomial algorithm for the linear problem with (I). Actually, if we replace $p(n)$ by a suitable polynomial of the size of the input, we obtain a polynomial algorithm for the general case. We will develop such an algorithm in Section 3. It is based on the general algorithm described in the next subsection.

2.5 General algorithm for finding inequalities $x_j \leq 1/2$

Now we consider a convex feasibility problem

$$x \in S \cap \mathbb{R}^n_+,$$

where S is a closed convex set in $\mathbb{R}^n$ with the property that

$$S \cap \mathbb{R}^n_+ \neq \emptyset \Rightarrow S \cap [0,1]^n \neq \emptyset.$$

That is, it is assumed that if the problem has a solution, then there is a solution in the unit cube $[0,1]^n$.

Consider a map $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ such that

$$\|\varphi(x) - y\| \leq \|x - y\|, \quad \forall x \in \mathbb{R}^n,\ \forall y \in S.$$

Additionally, we assume that

$$\varphi\left(\mathbb{R}^n\right) \subseteq S,$$

i.e., the codomain of φ is a subset of S. For example, the operator of orthogonal projection onto $\{x : Ax = b\}$ is such a map with respect to $\{x : Ax = b\}$. More generally, all non-expansive maps whose codomain is S and for which all vectors in S are fixed points are examples of such maps.

Let X be an $n \times n$ real matrix. Consider $\Phi : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ such that

$$\Phi(X) = (\varphi(X\mathbf{e}_1) \ldots \varphi(X\mathbf{e}_n)).$$

In other words, each column $\Phi(X)\mathbf{e}_j$ of $\Phi(X)$ is the image of the respective column $X\mathbf{e}_j$ of X under the map φ.

Let I denote the identity matrix and O denote an $n \times n$ zero matrix. The following algorithm either finds a solution in $S \cap \mathbb{R}^n_+$ or returns a non-empty set $J \subseteq \{1, \ldots, n\}$ such that

$$x_j \leq \frac{1}{2}, \quad \forall x \in S \cap [0,1]^n, \forall j \in J.$$

Algorithm 2.1

$X := \Phi(O)$;
$u := n\mathbf{1}$;
while $u \geq \mathbf{0}$ and $X\mathbf{1} \not\geq \mathbf{0}$
$\Delta := \max\left(X, \frac{1}{2}I\right) - X$;

$X := \Phi(X + \Delta)$;

$u := u - (\Delta \circ \Delta)^T \mathbf{1}$;

end

if $X\mathbf{1} \geq \mathbf{0}$ **then** return $\frac{1}{n} X\mathbf{1}$;

else return $J = \{j : u_j < 0\}$.

The $\circ$ denotes the Hadamard product. That is, $\Delta \circ \Delta$ is the matrix obtained from Δ by squaring the elements.

This algorithm generates a sequence of matrices in $\mathbb{R}^{n \times n}$. At each iteration, the algorithm checks whether $X\mathbf{1}$ is a nonnegative vector. If it is so, then $\frac{1}{n} X\mathbf{1}$ is a feasible solution because $\frac{1}{n} X\mathbf{1} \in S$ due to the fact that S is convex and each column of X belongs to S in the course of the algorithm due to the property that $\varphi(\mathbb{R}^n) \subseteq S$.

Let

$$Q_j = \left\{ x \in [0,1]^n : x_j \geq \frac{1}{2} \right\}.$$

The following lemma explains the role of the vector u.

Lemma 2.1 *For all iterations of the algorithm,*

$$u_j \geq \|X\mathbf{e}_j - x\|^2, \quad \forall x \in S \cap Q_j, \ \forall j.$$

Proof. The initial $u = n\mathbf{1}$ satisfies the above inequality because, for all $x \in S$,

$$\|\Phi(O)\mathbf{e}_j - x\|^2 \leq \|\mathbf{0} - x\|^2 \leq n, \quad \forall j.$$

Note that

$$\|(X + \Delta)\mathbf{e}_j - x\|^2 \leq \|X\mathbf{e}_j - x\|^2 - \mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j, \quad \forall x \in Q_j,$$

because $(X + \Delta)\mathbf{e}_j$ is obtained by the orthogonal projection of $X\mathbf{e}_j$ onto $\mathbb{R}^n_+ + \frac{1}{2}\mathbf{e}_j$. The above inequality follows from basic properties of orthogonal projections; see Section 2.6 for more details. Then

$$\|\Phi(X + \Delta)\mathbf{e}_j - x\|^2 \leq \|(X + \Delta)\mathbf{e}_j - x\|^2 \leq \|X\mathbf{e}_j - x\|^2 - \mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j, \quad \forall x \in S \cap Q_j.$$

If

$$u'_j \geq \|X\mathbf{e}_j - x\|^2, \ \forall x \in S \cap Q_j,$$

then

$$u'_j - \mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j \geq \|\Phi(X + \Delta)\mathbf{e}_j - x\|^2, \ \forall x \in S \cap Q_j.$$

This explains the formula for u in the algorithm. The lemma follows.

■

Now it should be clear that if $u_j < 0$ at some iteration, then $x_j \leq 1/2$ for all x in $S \cap [0,1]^n$. This follows from the above lemma. The algorithm returns $\frac{1}{n}X\mathbf{1}$ only if $\frac{1}{n}X\mathbf{1}$ is a feasible solution. Indeed, we have $\frac{1}{n}X\mathbf{1} \geq \mathbf{0}$ in this case. Moreover, $\frac{1}{n}X\mathbf{1} \in S$ because $\varphi(\mathbb{R}^n) \subseteq S$ and S is convex. Thus the output of the algorithm is correct.

We will use the following lemma to estimate the running time in Section 3.

Lemma 2.2 *Let $a \in \mathbb{R}^n$ and $\delta \in \mathbb{R}$. If $a^T\mathbf{1} \leq \delta$, then*

$$\left(\max\left(a, \frac{1}{2}\mathbf{e}_i\right) - a\right)^T \mathbf{1} \geq \frac{1}{2} - \delta$$

for all $i \in \{1, \ldots, n\}$.

Proof. $\left(\max\left(a, \frac{1}{2}\mathbf{e}_i\right) - a\right)^T \mathbf{1} \geq \left(\frac{1}{2}\mathbf{e}_i - a\right)^T \mathbf{1} \geq \frac{1}{2} - \delta.$ ■

To understand how this simple lemma works in the context of our algorithm, let us consider an iteration with $X\mathbf{1} \ngeq \mathbf{0}$. Let $a^T = \mathbf{e}_i^T X$ for some i with $\mathbf{e}_i^T X\mathbf{1} \leq 0$. We have $a^T\mathbf{1} \leq 0$. Note that

$$\left(\max\left(a, \frac{1}{2}\mathbf{e}_i\right) - a\right)^T = \mathbf{e}_i^T\Delta.$$

Lemma 2.2, applied to a defined above and $\delta = 0$, implies that

$$\frac{1}{2} \leq \mathbf{e}_i^T\Delta\mathbf{1} \leq \|\mathbf{e}_i^T\Delta\|\sqrt{n}.$$

It follows that

$$\mathbf{e}_i^T(\Delta \circ \Delta)\mathbf{1} = \|\mathbf{e}_i^T\Delta\|^2 \geq \frac{1}{4n}, \quad \forall i, \mathbf{e}_i^T X\mathbf{1} \leq 0.$$

Taking into account this inequality and the formula for u in the algorithm, together with the fact that the algorithm must terminate if some u_j becomes negative, we obtain

Theorem 2.2 *Algorithm 2.1 produces a correct output (either a vector x in $S \cap \mathbb{R}^n_+$ or a nonempty set J such that $x_j \leq 1/2$ for all x in $S \cap [0,1]^n$ and j in J in at most $4n^2$ iterations.*

The above general algorithm implies a polynomial-time algorithm for linear programming. To see this, we simply consider $Ax = b, x \geq \mathbf{0}$, where A and b are rational. Without loss of generality, we assume that if the system is feasible, then it has a solution in $[0,1]^n$. Let $S = \{x : Ax = b\}$. It is a well-known fact that the operator $\pi_{\{x:Ax=b\}}$ of orthogonal projection onto $\{x : Ax = b\}$ has all the properties that we require for φ. The projection of a vector on $\{x : Ax = b\}$ can be computed in polynomial time. Then, Theorem 2.2 implies that in polynomial time we can find a feasible solution or an index j such that $x_j \leq 1/2$ for all feasible solutions x in $[0,1]^n$. If a feasible solution is found, then we return this solution. Otherwise, we divide the jth column of A by 2 an repeat the above procedure. This division preserves the property that the feasibility problem has a solution in $[0,1]^n$. It should be clear that the same idea works also for linear optimization problems. Based on this general idea, in Section 3 we develop a polynomial algorithm for linear programming which is strongly polynomial for linear problems with (I).

2.6 Orthogonal projections

In this subsection, we discuss some well-known facts about orthogonal projections and introduce some additional notation.

The orthogonal projection of a vector v in a Euclidean space on a closed convex set S in this Euclidean space is the vector $\pi_S(v)$ in S such that

$$\|\pi_S(v) - v\| = \min\{\|w - v\| : w \in S\},$$

where $\|\cdot\|$ denotes the Euclidean norm. The following inequality is a well-known property of orthogonal projections:

$$\|\pi_S(v^0) - v\|^2 \leq \|v^0 - v\|^2 - \|\pi_S(v^0) - v^0\|^2, \quad \forall v \in S. \tag{2.1}$$

This inequality means that the projection brings v^0 closer to each point of S; see for example Boyd [4] and Dattorro [9] for further information on orthogonal projections.

Let S_1 and S_2 denote closed convex sets. Performing projections of v^0 first onto S_2 and then onto S_1, we obtain the point $\pi_{S_1}\pi_{S_2}(v^0)$. Inequality (2.1) implies

$$\|\pi_{S_1}\pi_{S_2}(v^0) - v\|^2 \leq \|v^0 - v\|^2 - \|\pi_{S_2}(v^0) - v^0\|^2 - \|\pi_{S_1}\pi_{S_2}(v^0) - \pi_{S_2}(v^0)\|^2, \quad \forall v \in S_1 \cap S_2.$$

Replacing v^0 by $\pi_{S_1}\pi_{S_2}(v^0)$, we reduce the distance from the current vector to each vector v in $S_1 \cap S_2$. (This is actually an iteration of the well-known alternating projection method; e.g., see Dattorro [9] and Cheney and Goldstein [5].) To measure the progress, in the sense of the reduction of the distance, we observe the following. If u_j is an upper bound on $\|v^0 - v\|^2$ for all $v \in S_1 \cap S_2$, then the above inequality implies

$$u_j - \|\pi_{S_2}(v^0) - v^0\|^2 - \|\pi_{S_1}\pi_{S_2}(v^0) - \pi_{S_2}(v^0)\|^2 \geq \|\pi_{S_1}\pi_{S_2}(v^0) - v\|^2, \quad \forall v \in S_1 \cap S_2.$$

In order to find a point in $S_1 \cap S_2$ or somewhere close to this intersection, we can undertake the following steps. First of all, we observe that if the new bound, i.e., the value at the left-hand side of the above inequality, is negative, then $S_1 \cap S_2$ is empty. In this case we simply stop our procedure. Otherwise, we replace u_j by the new bound and go further. Of course, we can use this approach to measure the progress of the alternating projections, but, in its pure form, when we perform iterations in the original space, this approach does not lead to a polynomial algorithm. However, as we will see, this simple idea leads to a polynomial algorithm for linear programming if we consider n vectors simultaneously and apply an appropriate procedure to choose the sets onto which these vectors should be projected. The set of these vectors is represented in the form of a matrix. That is, unlike traditional algorithms for linear programming, which generate sequences of solutions in the original vector space, our algorithm generates a sequence in a vector space of matrices.

It remains to discuss some technical details concerning orthogonal projections. Let

$$P = I - A^T(AA^T)^{-1}A,$$

where I is the $n \times n$ identity matrix. This is the matrix of the orthogonal projection onto the linear subspace $\{x : Ax = \mathbf{0}\}$. Let

$$H = \{x \in \mathbb{R}^n : Ax = b\}.$$

The orthogonal projection on H is computed as

$$\pi_H(x) = Px + A^T(AA^T)^{-1}b.$$

Let ξ be an upper bound on the optimal value of our linear optimization problem. The set

$$H(\xi) = \{x \in \mathbb{R}^n : Ax = b, c^T x \leq \xi\}$$

is the set of solutions of $Ax = b$ with $c^T x \leq \xi$. If $Pc = \mathbf{0}$ and $H(\xi) \neq \emptyset$, then $H = H(\xi)$. If $x \in H$, then

$$\pi_{H(\xi)}(x+y) = \begin{cases} x + Py - \dfrac{\max(0, c^T(x+Py) - \xi)}{\|Pc\|^2} Pc, & \text{if } Pc \neq \mathbf{0}, \\ x + Py, & \text{otherwise.} \end{cases}$$

for every vector y. Note that

$$x + Py = \pi_H(x + y),$$

provided that $x \in H$. That is, $\pi_{H(\xi)}(x+y)$ is obtained by projecting $x+y$ onto H and then onto the halfspace defined by the inequality

$$c^T Px \leq \xi - c^T A^T(AA^T)^{-1}b,$$

which is satisfied by x in H if and only if $c^T x \leq \xi$. (To see this, we use the fact that $x = \pi_H(x)$ for all x in H.)

Let $\max(\cdot,\cdot)$ denote the componentwise maximum. Let X be an $n \times n$ real matrix and D be an $n \times n$ 0-1 diagonal matrix. Consider an $n \times n$ *nonnegative* matrix Δ such that

$$\Delta \leq \max\left(XD, \frac{1}{2}D\right) - XD. \tag{2.2}$$

Note that if $D_{jj} = 0$ then $\Delta \mathbf{e}_j = \mathbf{0}$. Moreover,

$$\max\left(XD, \frac{1}{2}D\right) = \pi_{\mathbb{R}_+^{n\times n} + \frac{1}{2}D}(XD).$$

That is, $\max\left(XD, \frac{1}{2}D\right)$ is the projection of XD on $\mathbb{R}_+^{n\times n} + \frac{1}{2}D$ in the sense of the Euclidean norm on $\mathbb{R}^{n\times n}$. If $\Delta = \max\left(XD, \frac{1}{2}D\right) - XD$, then, for all j with $D_{jj} = 1$,

$$(X + \Delta)\mathbf{e}_j = \pi_{\mathbb{R}_+^n + \frac{1}{2}\mathbf{e}_j}(X\mathbf{e}_j).$$

Then, for all $j = 1, \ldots, n$,

$$\|(XD + \Delta)\mathbf{e}_j - x\|^2 \leq \|XD\mathbf{e}_j - x\|^2 - \mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j, \quad \forall x \in \mathbb{R}^n_+ + \frac{1}{2}\mathbf{e}_j, \tag{2.3}$$

where $\circ$ denotes the Hadamard product of two matrices, which is simply their componentwise product. The value $\mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j$ is equal to $\sum_{i=1}^n (\Delta_{ij})^2$, which is the sum of squares of the elements of Δ in the jth column. Inequality (2.3) follows from (2.1). To see this, we set

$$v^0 := XD\mathbf{e}_j$$

and

$$S := \mathbb{R}^n_+ + \frac{1}{2}D\mathbf{e}_j,$$

Let us prove that the inequality (2.3) is true also in the general case of (2.2). Denote

$$V = \frac{1}{2}D - \left(\max\left(XD, \frac{1}{2}D\right) - XD - \Delta\right).$$

Again, (2.3) follows from (2.1). To show this, we define v^0 as before and S as

$$S := \mathbb{R}^n_+ + V\mathbf{e}_j.$$

Note that

$$\mathbb{R}^n_+ + \frac{1}{2}D\mathbf{e}_j \subseteq S.$$

and

$$(XD + \Delta)\mathbf{e}_j = \pi_S(XD\mathbf{e}_j),$$

which implies (2.3) by (2.1) and can be proved as follows. Consider $i = 1, \ldots, n$, and the following two cases:

1. $\mathbf{e}_i^T \max\left(XD, \frac{1}{2}D\right)\mathbf{e}_j = \mathbf{e}_i^T XD\mathbf{e}_j$. In this case $\mathbf{e}_i^T \Delta \mathbf{e}_j = 0$ and

$$\frac{1}{2}\mathbf{e}_i^T D\mathbf{e}_j = \mathbf{e}_i^T V\mathbf{e}_j.$$

   Then

$$\mathbf{e}_i^T(XD + \Delta)\mathbf{e}_j = \mathbf{e}_i^T XD\mathbf{e}_j = \mathbf{e}_i^T \max\left(XD, \frac{1}{2}D\right)\mathbf{e}_j = \mathbf{e}_i^T \max\left(XD, V\right)\mathbf{e}_j.$$

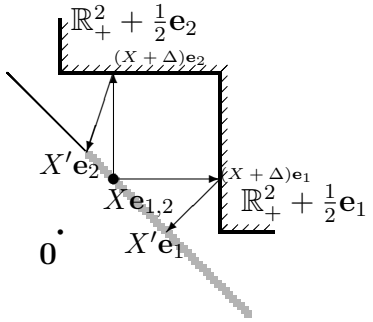

Figure 1: $H(\frac{1}{6}) = \{x : x_1 + x_2 = \frac{1}{3}, -x_1 + x_2 \leq \frac{1}{6}\}$; $X\mathbf{e}_j = X\mathbf{e}_2 = \pi_{H(\xi)}(\mathbf{0})$.

2. $\mathbf{e}_i^T \max\left(XD, \frac{1}{2}D\right)\mathbf{e}_j = \frac{1}{2}\mathbf{e}_i^T D\mathbf{e}_j$. In this case

$$\mathbf{e}_i^T(XD + \Delta)\mathbf{e}_j = \mathbf{e}_i^T V\mathbf{e}_j.$$

Then, since Δ is a nonnegative matrix,

$$\mathbf{e}_i^T(XD + \Delta)\mathbf{e}_j = \mathbf{e}_i^T(\max(XD, XD + \Delta)\mathbf{e}_j = \mathbf{e}_i^T \max\left(XD, V\right)\mathbf{e}_j.$$

It follows that

$$(XD + \Delta)\mathbf{e}_j = \max\left(XD, V\right)\mathbf{e}_j = \max\left(XD\mathbf{e}_j, V\mathbf{e}_j\right) = \pi_{\mathbb{R}^n_+ + V\mathbf{e}_j}(XD\mathbf{e}_j).$$

Thus, using inequalities (2.1) and (2.3), for each j with $D_{jj} = 1$ we can write

$$\begin{aligned} \|\pi_{H(\xi)}((X + \Delta)\mathbf{e}_j) - x\|^2 &\leq \|(X + \Delta)\mathbf{e}_j - x\|^2 \\ &\leq \|X\mathbf{e}_j - x\|^2 - \|\Delta\mathbf{e}_j\|^2, \quad \forall x \in H(\xi) \cap \left(\mathbb{R}^n_+ + \tfrac{1}{2}\mathbf{e}_j\right). \end{aligned} \tag{2.4}$$

The example in Figure 1 illustrates the projections discussed above. In this example, $\xi = \frac{1}{6}$, $(A|b) = (1 \quad 1 \mid 1/3)$, and $c^T = (-1, 1)$. The matrix X is equal to $(\pi_{H(\xi)}(\mathbf{0}), \pi_{H(\xi)}(\mathbf{0}))$. That is, X consists of two columns, each of which is equal to $\pi_{H(\xi)}(\mathbf{0})$ depicted as a large black circle in the picture. The matrix Δ is computed as $\Delta = \max\left(XD, \frac{1}{2}D\right) - XD$, where matrix D is the 2×2 identity matrix. The matrix X' is equal to $(\pi_{H(\xi)}((X+\Delta)\mathbf{e}_1), \pi_{H(\xi)}((X+\Delta)\mathbf{e}_2))$. That is, $X'\mathbf{e}_j = \pi_{H(\xi)}((X + \Delta)\mathbf{e}_j)$ for $j = 1, 2$. The gray segment depicts $H(\xi)$. The thin line, partially covered by the gray segment, is $\{x : Ax = b\}$.

3 Polynomial algorithm for linear optimization

We continue to consider the problem

$$\min\{c^T x : Ax = b, x \geq \mathbf{0}\}.$$

Now we assume that

(II) if the problem is feasible then there exists an optimal solution x^* such that $x^*_j \in \{0\} \cup [\beta_j, 1]$ for all $j = 1, \ldots, n$,

where the values β_j in $(0, 1]$ are known in advance. All optimal solutions with the above property lie in the unit cube $[0, 1]^n$. This implies that

$$LB = -\|Pc\|_1 + c^T A^T (AA^T)^{-1} b$$

is a lower bound on the optimal value and

$$UB = \|Pc\|_1 + c^T A^T (AA^T)^{-1} b$$

is an upper bound on the optimal value. Here, $\|\cdot\|_1$ denotes the 1-norm. To obtain the above bounds, we consider an optimal solution x^* in $[0, 1]^n$ and take into account $x^* = \pi_H(x^*)$. This implies $c^T x^* = c^T P x^* + c^T A^T (AA^T)^{-1} b$ and the above bounds.

In this section we develop an algorithm whose running time is bounded by a polynomial in n and in the values $\left\lceil \log \frac{2}{\beta_j} \right\rceil$. This implies a polynomial running time in the general case when A, b, and c are rational because any linear problem can be represented in the form with the property (II) by means of standard tools in polynomial time. In the case of a problem with the property (I), we obtain a strongly polynomial algorithm.

3.1 Basic idea

In this subsection, we introduce some additional notation and explain the main idea of the algorithm.

Iteration k of our algorithm considers a linear problem

$$\begin{aligned} \min \quad & c^T M^{(k)} x \\ \text{s.t.} \quad & A^{(k)} x = b^{(k)}, \\ & x \geq \mathbf{0}, \end{aligned}$$

further denoted as $LP^{(k)}$, with $A^{(k)}$ of full rank. This problem is a reformulation of

$$\begin{aligned} \min \quad & c^T M^{(k)} x \\ \text{s.t.} \quad & A M^{(k)} x = b, \\ & (I - D^{(k)}) x = \mathbf{0}, \\ & x \geq \mathbf{0}, \end{aligned}$$

where $M^{(k)}$ is a positive definite diagonal matrix and $D^{(k)}$ is a 0-1 diagonal matrix such that $D^{(k)}_{jj} = 0$ only if $x^*_j = 0$ for all optimal solutions x^*, with (II), of the original problem. If $D^{(k)}_{jj} = 0$, then $x_j = 0$ for each feasible solution x of $LP^{(k)}$.

We define $LP^{(0)}$ as the original problem with (II). Let $D^{(0)} = I$ and $M^{(0)} = I$.

The optimal value of $LP^{(k)}$ is equal to the optimal value of the original problem. If x is a feasible solution of $LP^{(k)}$, then $M^{(k)}x$ is a feasible solution of the original problem. Since the optimal values coincide, $M^{(k)}x$ is optimal for the original problem if x is optimal for $LP^{(k)}$; see Lemma 3.1 for details.

Let OPT denote the optimal value of the original problem. In view of the above remarks, OPT is at the same time the optimal value for every problem $LP^{(k)}$. Further, optimal solutions of the original problem are called optimal solutions.

At each iteration, our algorithm preserves the property that each problem $LP^{(k)}$ has an optimal solution in $[0,1]^n$. Then, in the same way as for the original problem,

$$LB^{(k)} = -\|P^{(k)} M^{(k)} c\|_1 + c^T M^{(k)} (A^{(k)})^T (A^{(k)} (A^{(k)})^T)^{-1} b^{(k)}$$

is a lower bound on OPT and

$$UB^{(k)} = \|P^{(k)} M^{(k)} c\|_1 + c^T M^{(k)} (A^{(k)})^T (A^{(k)} (A^{(k)})^T)^{-1} b^{(k)}$$

is an upper bound on OPT, where

$$P^{(k)} = I - (A^{(k)})^T (A^{(k)} (A^{(k)})^T)^{-1} A^{(k)}.$$

So we will use $^{(k)}$ to emphasize that the respective object is constructed at iteration k. In particular, we define

$$H^{(k)}(\xi) := \{x \in \mathbb{R}^n : A^{(k)}x = b^{(k)}, c^T M^{(k)} x \leq \xi\},$$

where $\xi \geq OPT$.

The main goal of iteration k of the algorithm is to construct a new matrix $X^{(k+1)}$ such that $X^{(k+1)}\mathbf{e}_j \in H^{(k+1)}(\xi)$ for all j with $D^{(k+1)}_{jj} = 1$ from the current matrix $X^{(k)}$ whose columns with $D^{(k)}_{jj} = 1$ belong to $H^{(k)}(\xi)$ by means of some rule which guarantees a progress in solving the original problem. Let us consider

$$\bar{x} = \frac{1}{\sum_{i=1}^n D^{(k)}_{ii}} X^{(k)} D^{(k)} \mathbf{1},$$

which is the centroid of those columns of $X^{(k)}$ that correspond to nonzero diagonal elements of $D^{(k)}$. It is clear that $c^T M^{(k)} \bar{x} \leq \xi$ and $A^{(k)}\bar{x} = b^{(k)}$. If $\bar{x} \geq \mathbf{0}$, then $\bar{x}$ is a feasible solution of $LP^{(k)}$ with $c^T M^{(k)} \bar{x} \leq \xi$. Note that $A^{(k)}\bar{x} = b^{(k)}$ implies $\bar{x}_i = 0$ for all i such that $D^{(k)}_{ii} = 0$.

Let $\gamma = \frac{1}{4}$. If $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} > \gamma$ for all i with $D^{(k)}_{ii} = 1$, then $\bar{x}$ is feasible for $LP^{(k)}$ and $c^T M^{(k)} \bar{x}$ is an upper bound on OPT. This upper bound does not exceed ξ. In this case, the algorithm improves the current upper bound ξ by at least $\frac{\gamma}{n^2} \|P^{(k)} M^{(k)} c\|_1$. The algorithm uses the fact that if the components $\bar{x}_i$ with $D^{(k)}_{ii} = 1$ are sufficiently large (greater than γ/n), then there is a simple procedure that transforms $\bar{x}$ into another feasible solution of $LP^{(k)}$ with the objective value not exceeding $c^T M^{(k)} \bar{x} - \frac{\gamma}{n^2} \|P^{(k)} M^{(k)} c\|_1$. As we will see, the number of such improvements is polynomially bounded.

Now let $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma$ for some i with $D^{(k)}_{ii} = 1$. Then the set

$$T = \left\{ i : \mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma,\ D^{(k)}_{ii} = 1 \right\} \cup \left\{ i : \exists j, \mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{e}_j \leq -1 \right\}.$$

is not empty. This set contains all indices i with $D^{(k)}_{ii} = 1$ such that $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma$ or some element in the ith row of the matrix $X^{(k)} D^{(k)}$ does not exceed -1. Now we consider

$$\Delta = \text{diag } \left(\sum_{i \in T} \mathbf{e}_i \right) \left(\max \left(X^{(k)} D^{(k)}, \frac{1}{2} \cdot D^{(k)} \right) - X^{(k)} D^{(k)} \right).$$

Actually, to obtain a polynomial running time, it would be sufficient to simply consider the identity matrix in place of diag $\left(\sum_{i \in T} \mathbf{e}_i \right)$. The only reason to consider T is that some

negative elements of $X^{(k)}D^{(k)}$ can be too small in absolute value to be useful in the progress of the algorithm. Using the multiplication by diag $\left(\sum_{i\in T}\mathbf{e}_i\right)$, we save a factor of n in the future estimate of the running time.

To measure the progress of the algorithm after the above step, we will use the properties of orthogonal projections discussed in Section 2.6. Denote

$$Q_j = \left\{x \in [0,1]^n : x_j \geq \frac{1}{2}\right\}.$$

This set is a half of the unit cube $[0,1]^n$. If $H^{(k)}(\xi) \cap Q_j$ is empty, then $x^*_j < \frac{1}{2}$ for all optimal solutions x^* of $LP^{(k)}$ in $[0,1]^n$. Let $u^{(k)}$ in $\mathbb{R}^n$ be such that

$$u^{(k)}_j \geq \|X^{(k)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k)}(\xi) \cap Q_j,\ \forall j, D^{(k)}_{jj} = 1.$$

In the fashion of the general algorithm of Section 2.5, we consider

$$u := u^{(k)} - (\Delta \circ \Delta)^T\mathbf{1}.$$

Vector u is obtained by subtracting $\mathbf{1}^T(\Delta \circ \Delta)\mathbf{e}_j = \sum_{i=1}^n(\Delta_{ij})^2$, i.e., the sum of squares of elements of Δ in the jth *column* of Δ, from each component $u^{(k)}_j$. Note that $u_j = u^{(k)}_j$ for all j with $D^{(k)}_{jj} = 0$ because $\Delta\mathbf{e}_j = \mathbf{0}$ in this case. Matrix Δ satisfies (2.2) (see Section 2.6) for $X = X^{(k)}$ and $D = D^{(k)}$. Since $Q_j \subset \mathbb{R}^n_+ + \frac{1}{2}\mathbf{e}_j$, inequality (2.4) implies

$$u_j \geq \|(X^{(k)} + \Delta)\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k)}(\xi) \cap Q_j,\ \forall j, D^{(k)}_{jj} = 1.$$

If $u_j < 0$ for some j with $D^{(k)}_{jj} = 1$, then $H^{(k)}(\xi) \cap Q_j$ is empty. Then $x^*_j < \frac{1}{2}$ for all optimal solutions x^* of $LP^{(k)}$ in $[0,1]^n$. In this case the algorithm scales the current problem $LP^{(k)}$. Then we need some correction of u to construct the next vector $u^{(k+1)}$. Also, the computation of $X^{(k+1)}$ in this case needs some more efforts than in the case $u \geq \mathbf{0}$. The details are given in the algorithm and in the proof of Lemma 3.1.

If $u \geq \mathbf{0}$, then iteration k does not perform any scaling and does not add new equations, i.e, $M^{(k+1)} = M^{(k)}$ and $D^{(k+1)} = D^{(k)}$, which means $H^{(k+1)}(\xi) = H^{(k)}(\xi)$. Then the algorithm sets

$$X^{(k+1)}\mathbf{e}_j := \pi_{H^{(k+1)}(\xi)}((X^{(k)} + \Delta)\mathbf{e}_j), \quad \forall j, D^{(k+1)}_{jj} = 1,$$

and

$$u^{(k+1)} := u.$$

Then we have

$$\|(X^{(k)} + \Delta)\mathbf{e}_j - x\|^2 \geq \|X^{(k+1)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k+1)}(\xi) \cap Q_j, \ \forall j, D_{jj}^{(k+1)} = 1.$$

Therefore,

$$u_j^{(k+1)} \geq \|X^{(k+1)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k+1)}(\xi) \cap Q_j, \ \forall j, D_{jj}^{(k+1)} = 1.$$

By our current assumption, $\bar{x}_i \leq \gamma$ for some i with $D_{ii}^{(k)} = 1$. As we will show later when analyzing the running time of the algorithm, this implies that the value $\mathbf{e}_i^T(\Delta \circ \Delta)\mathbf{1} = \sum_{j=1}^{n}(\Delta_{ij})^2$, i.e., the sum of squares of elements of Δ in the ith *row*, is not less than some positive value depending on the dimension. Since

$$\mathbf{1}^T u^{(k+1)} \leq \mathbf{1}^T u^{(k)} - \mathbf{1}^T(\Delta \circ \Delta)^T\mathbf{1}$$

for any iteration with $M^{(k+1)} = M^{(k)}$, it follows that, starting with any iteration, we reach an iteration with $u_j < 0$ in a polynomial number of iterations. This implies a polynomial bound on the number of iterations in any sequence of consecutive iterations with $M^{(k+1)} = M^{(k)}$. On the other hand, as we will see later, the number of iterations with $M^{(k+1)} \neq M^{(k)}$ is bounded by a polynomial in the values $\lceil \log \frac{2}{\beta_j} \rceil$ due to (II). This leads to a polynomial running time.

3.2 Algorithm

Let O denote an $n \times n$ zero matrix. The arrow $\leftarrow$ will mean that $(A'|b')$ is mapped to $(A''|b'')$ of full rank such that the rows of $(A''|b'')$ span the same space as the rows of $(A'|b')$. Note that such a matrix $(A''|b'')$ can be constructed in $O(n^3)$ time in such a way that the binary size of $(A''|b'')$ is polynomially bounded in the size of $(A'|b')$ in the case of rational coefficients.

Now we will give a complete description of the algorithm.

Algorithm 3.1

Input: A linear optimization problem with (II);

Output: An optimal solution x^* or a decision that the problem is infeasible.

$k := 0$;

$\xi := UB$;

$X^{(0)} := \pi_{L(\xi)}(O)$;

$u^{(0)} := n\mathbf{1}$;

$H^{(0)}(\xi) := H(\xi)$;

$D^{(0)} := I$;

$M^{(0)} := I$;

$\gamma := \frac{1}{4}$;

repeat

while $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma$ for some i with $D_{ii}^{(k)} = 1$

1. $T := \left\{ i : \mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma,\ D_{ii}^{(k)} = 1 \right\} \cup \left\{ i : \exists j, \mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{e}_j \leq -1 \right\}$;
   $\Delta := \text{diag } \left(\sum_{i \in T} \mathbf{e}_i \right) \left(\max \left(X^{(k)} D^{(k)}, \frac{1}{2} \cdot D^{(k)} \right) - X^{(k)} D^{(k)} \right)$;
2. $u := u^{(k)} - (\Delta \circ \Delta)^T \mathbf{1}$;
3. For each j, $M_{jj}^{(k+1)} := \begin{cases} \frac{M_{jj}^{(k)}}{2}, & \text{if } u_j < 0, D_{jj}^{(k)} = 1, \\ M_{jj}^{(k)}, & \text{otherwise.} \end{cases}$
4. For each j, $D_{jj}^{(k+1)} := \begin{cases} 0, & \text{if } \beta_j > M_{jj}^{(k+1)}, \\ D_{jj}^{(k)}, & \text{otherwise.} \end{cases}$

   if $D^{(k+1)} = O$ **then** return $x^* = \mathbf{0}$ if $\mathbf{0}$ is feasible and "no solutions", otherwise;
5. $(A^{(k+1)} | b^{(k+1)}) \leftarrow \left(\begin{array}{c|c} AM^{(k+1)} & b \\ I - D^{(k+1)} & \mathbf{0} \end{array} \right)$;
6. **if** rank $(A^{(k+1)} | b^{(k+1)}) \neq$ rank $A^{(k+1)}$ **then** return "no solutions";
7. For each j, $X^{(k+1)} \mathbf{e}_j := \begin{cases} \pi_{H^{(k+1)}(\xi)}((X^{(k)} + \Delta)\mathbf{e}_j), & \text{if } M_{jj}^{(k+1)} = M_{jj}^{(k)}, D_{jj}^{(k+1)} = 1, \\ X^{(k)} \mathbf{e}_j, & \text{if } D_{jj}^{(k+1)} = 0, \\ \pi_{H^{(k+1)}(\xi)}(\mathbf{0}), & \text{otherwise.} \end{cases}$
8. For each j, $u_j^{(k+1)} := \begin{cases} u_j + 4|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|, & \text{if } M_{jj}^{(k+1)} = M_{jj}^{(k)}, \\ n, & \text{otherwise.} \end{cases}$

9. $k := k + 1$;

end

if $P^{(k)}M^{(k)}c \neq \mathbf{0}$

$\xi := \min\left\{UB^{(k)}, \xi - \frac{\gamma}{n^2}\|P^{(k)}M^{(k)}c\|_1\right\}$;

$$X^{(k+1)}\mathbf{e}_j := \begin{cases} \pi_{H^{(k)}(\xi)}(X^{(k)}\mathbf{e}_j), & D_{jj}^{(k+1)} = 1, \\ X^{(k)}\mathbf{e}_j, & D_{jj}^{(k+1)} = 0. \end{cases}$$

$(A^{(k+1)}|b^{(k+1)}) := (A^{(k)}|b^{(k)})$;

$D^{(k+1)} := D^{(k)}$;

$M^{(k+1)} := M^{(k)}$;

$u^{(k+1)} := u^{(k)}$;

$k := k + 1$;

end

until $P^{(k)}M^{(k)}c = \mathbf{0}$

Return $x^* = \frac{1}{\sum_{i=1}^n D_{ii}^{(k)}} M^{(k)}X^{(k)}D^{(k)}\mathbf{1}$;

Remark. We call all steps between two consecutive increments of k an *iteration* and assign the respective value of k to each iteration. So we have iterations $k = 0, 1, \ldots$.

Remark. At steps 3 and 4, we assume that the non-diagonal elements of $M^{(k+1)}$ and $D^{(k+1)}$ are set to 0.

Remark. We could use elimination of columns in place of multiplications by $D^{(k)}$. However, it seems to be more convenient to use $D^{(k)}$ because then the dimension remains the same. So the only role of $D^{(k)}$ is to "switch off" the columns that we do not need anymore.

In the formulation and in the proof of the following lemma we use the notation introduced in Section 3.1. Again, optimal solutions of the original problem with (II) will be simply called optimal solutions.

Lemma 3.1 *At iteration k of the algorithm:*

(a) For all j with $D_{jj}^{(k+1)} = 1$,

$$u_j^{(k+1)} \geq \|X^{(k+1)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k+1)}(\xi) \cap Q_j. \tag{3.1}$$

(b) $(M^{(k+1)})^{-1}x^*$ *is an optimal solution of* $LP^{(k+1)}$ *for every optimal solution* x^* *with (II). Moreover,* $(M^{(k+1)})^{-1}x^* \in [0,1]^n$.

(c) $D_{jj}^{(k+1)} = 0 \Rightarrow x_j^* = 0$ *for all optimal solutions* x^* *with (II).*

(d) The value ξ *is an upper bound on the optimal value if the problem with (II) is feasible.*

Proof. Note that

$$u_j^{(0)} \geq \|X^{(0)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(0)}(\xi) \cap Q_j, \quad \forall j, D_{jj}^{(0)} = 1.$$

(Recall that $LP^{(0)}$ denotes the original problem. The $H^{(0)}(\xi)$ denotes $H(\xi)$.) Moreover, (b), (c), and (d) hold for 0 in place of $k+1$ because $M^{(0)}$ and $D^{(0)}$ are identity matrices. The initial ξ is equal to UB, which is an upper bound on OPT.

Assume that

$$u_j^{(l)} \geq \|X^{(k)}\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(l)}(\xi) \cap Q_j, \ \forall j, D_{jj}^{(l)} = 1,$$

for all $l = 0, \ldots, k$. In the same way, assume that (b),(c), and (d) hold when $k+1$ is replaced by $l = 0, \ldots, k$. These assumptions are our induction hypothesis.

If the condition of the inner loop is not satisfied, iteration k computes a new value of ξ. Then $(A^{(k+1)}|b^{(k+1)}) = (A^{(k)}|b^{(k)})$, $M^{(k+1)} = M^{(k)}$, and $D^{(k+1)} = D^{(k)}$, which implies (b) and (c) in this case.

Now assume that the condition of the inner loop is satisfied and iteration k performs steps 1-9. Let us analyze each step.

1. The matrix Δ is the product of the diagonal matrix diag $\left(\sum_{i\in T}\mathbf{e}_i\right)$ and the difference between the projection of $X^{(k)}D^{(k)}$ on $\mathbb{R}_+^{n\times n} + \frac{1}{2}D^{(k)}$ and $X^{(k)}D^{(k)}$. For all $i \notin T$, the ith row of Δ is zero. At step 7 we will use Δ to construct $X^{(k+1)}$.

2. The analysis given in Section 3.1 implies

$$u_j \geq \|(X^{(k)} + \Delta)\mathbf{e}_j - x\|^2, \quad \forall x \in H^{(k)}(\xi) \cap Q_j, \quad \forall j, D_{jj}^{(k)} = 1. \tag{3.2}$$

   The reason why we do not immediately compute $u^{(k+1)}$ is that the current problem $LP^{(k)}$ can be rescaled later in the algorithm, which can imply some correction. See step 8 for the computation of $u^{(k+1)}$.

3. This step constructs matrix $M^{(k+1)}$. Consider j with $D^{(k)}_{jj} = 1$. Since u_j satisfies the above inequality, $u_j < 0$ means that $H^{(k)}(\xi) \cap Q_j$ is empty. In this case,

$$x_j \leq 1/2$$

for all optimal solutions $x \in [0,1]^n$ of $LP^{(k)}$. Thus, if $u_j < 0$ and $D^{(k)}_{jj} = 1$, then the algorithm sets $M^{(k+1)}_{jj} := M^{(k)}_{jj}/2$. Otherwise, $M^{(k+1)}_{jj} := M^{(k)}_{jj}$.

Consider an optimal solution x^* with (II). By the induction hypothesis, $(M^{(k)})^{-1}x^*$ is an optimal solution of $LP^{(k)}$ and $(M^{(k)})^{-1}x^* \in [0,1]^n$. From the above inequality $x_j \leq 1/2$ it follows that $\mathbf{e}_j^T(M^{(k)})^{-1}x^* \leq 1/2$ for all j with $M^{(k+1)}_{jj} \neq M^{(k)}_{jj}$. Then

$$(M^{(k+1)})^{-1}x^* \in [0,1]^n.$$

4. **Proof of (b) and (c).** Let $\beta_j > M^{(k+1)}_{jj}$. We now prove that in this case x^*_j must be equal to 0 for all optimal solutions x^* with (II). Indeed, as shown above, $(M^{(k+1)})^{-1}x^* \in [0,1]^n$, which is possible only if $x^*_j = 0$ because otherwise we had $\mathbf{e}_j^T(M^{(k+1)})^{-1}x^* = x^*_j/M^{(k+1)}_{jj} \geq \beta_j/M^{(k+1)}_{jj} > 1$. So the algorithm sets $D^{(k+1)}_{jj} := 0$. That is, the column j of the coefficient matrix is "switched off". Step 5 adds the respective equations $x_j = 0$ to the constraints of the current problem. By the induction hypothesis, $D^{(k)}_{jj} = 0$ implies $x^*_j = 0$. So we have proved that $x^*_j = 0$ for all j with $D^{(k+1)}_{jj} = 0$. Then $(M^{(k+1)})^{-1}x^*$ is feasible for $LP^{(k+1)}$. On the other hand, let x^{**} be an optimal solution of $LP^{(k+1)}$. Then $M^{(k+1)}x^{**}$ is feasible for the original problem. This implies

$$c^T M^{(k+1)}x^{**} \geq c^T x^* = c^T M^{(k+1)}(M^{(k+1)})^{-1}x^*.$$

It follows that $(M^{(k+1)})^{-1}x^*$ is optimal for $LP^{(k+1)}$ and the optimal value of $LP^{(k+1)}$ is equal to the optimal value of the original problem. So we have proved (b) and (c).

It remains to note that if $D^{(k+1)} = O$, then, by (b) and (c), any optimal solution with (II) must be a zero vector. In this case the algorithm returns $x^* = \mathbf{0}$ if $\mathbf{0}$ is feasible. Otherwise, the algorithm decides that the problem is infeasible.

5. This step computes $A^{(k+1)}$ and $b^{(k+1)}$ for $LP^{(k+1)}$. If $M^{(k+1)} = M^{(k)}$ and $D^{(k+1)} = D^{(k)}$, then $LP^{(k+1)} = LP^{(k)}$. As shown above, the optimal value of $LP^{(k+1)}$ is equal to the optimal value of the original problem.

6. If the rank of the new augmented coefficient matrix is not equal to the rank of the new coefficient matrix, then there are no feasible solutions and the algorithm terminates.

7. This step computes $X^{(k+1)}$. For each j, the column $X^{(k+1)}\mathbf{e}_j$ is chosen depending on the conditions satisfied by j.

8. Now we prove that $u^{(k+1)}$ constructed at step 8 possesses the property **(a)**, i.e., that it satisfies (3.1).

   Case 1. Consider j with $M_{jj}^{(k+1)} = M_{jj}^{(k)}$ and $D_{jj}^{(k+1)} = 1$.

   Let x' be feasible for $LP^{(k+1)}$ and belong to $H^{(k+1)}(\xi) \cap Q_j$. Note that $D_{ii}^{(k)} = 0 \Rightarrow D_{ii}^{(k+1)} = 0$ for all i. Then the solution

$$x = (M^{(k)})^{-1} M^{(k+1)} x'$$

   is feasible for $LP^{(k)}$ and belongs to $H^{(k)}(\xi) \cap Q_j$. This solution is obtained from x' by dividing $|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|$ components of x' by 2. That is,

$$\mathbf{0} \leq x \leq x' \leq \mathbf{1}.$$

   Then

$$\|x'\|^2 - \|x\|^2 = (x' - x)^T (x' + x) \leq 2(x' - x)^T \mathbf{1} \leq 2|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|.$$

   Let $y = (X^{(k)} + \Delta)\mathbf{e}_j$. We have $y \geq -\mathbf{1}$ because $\mathbf{e}_i^T (X^{(k)} + \Delta)\mathbf{e}_j \geq 0$ for all i with $\mathbf{e}_i^T X^{(k)} \mathbf{e}_j \leq -1$ due to the fact that

$$\left\{i : \mathbf{e}_i^T X^{(k)} \mathbf{e}_j \leq -1\right\} \subset T.$$

   Then, taking into account $x \leq x'$, we write

$$y^T (x - x') = -y^T (x' - x) \leq \mathbf{1}^T (x' - x) \leq |\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|.$$

It follows that

$$\|(X^{(k)}+\Delta)\mathbf{e}_j - x'\|^2 = \|y\|^2 - 2y^T x' + \|x'\|^2$$

$$= \|y\|^2 - 2y^T x + \|x\|^2 + 2y^T(x-x') + \|x'\|^2 - \|x\|^2$$

$$= \|y-x\|^2 + 2y^T(x-x') + \|x'\|^2 - \|x\|^2 \leq \|y-x\|^2 + 4|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|$$

$$\leq u_j + 4|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|.$$

The latter inequality follows from (3.2). For every $x' \in H^{(k+1)}(\xi)$, the property (2.1) of orthogonal projections implies that

$$\|\pi_{H^{(k+1)}(\xi)}((X^{(k)}+\Delta)\mathbf{e}_j) - x'\| \leq \|(X^{(k)}+\Delta)\mathbf{e}_j - x'\|.$$

Therefore, we obtain (3.1) for

$$u_j^{(k+1)} = u_j + 4|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|.$$

Case 2. Now consider j with $M_{jj}^{(k+1)} \neq M_{jj}^{(k)}$ and $D_{jj}^{(k+1)} = 1$. In this case $X^{(k+1)}\mathbf{e}_j$ is the projection of $\mathbf{0}$ on $H^{(k+1)}(\xi)$. This implies (3.1) for $u_j^{(k+1)} = n$ because the the distance from $\mathbf{0}$ to any point in $H^{(k+1)}(\xi) \cap Q_j$ is not greater than $\sqrt{n}$.

9. Increase the counter.

Proof of (d). The initial $\xi = UB$ satisfies (d) if the problem with (II) is feasible because then there are optimal solutions in $[0,1]^n$. Consider the step computing ξ in the outer loop. If a new value of ξ is computed, then $D^{(k)} \neq O$ and $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} > \gamma$ for all i with $D_{ii}^{(k)} = 1$. Consider

$$\bar{x} = \frac{1}{\sum_{i=1}^n D_{ii}^{(k)}} X^{(k)} D^{(k)} \mathbf{1}.$$

Note that $\bar{x}_i > \frac{\gamma}{n} > 0$ for all i with $D_{ii}^{(k)} = 1$, because $\gamma > 0$, and $\bar{x}_i = 0$ for all i with $D_{ii}^{(k)} = 0$, which follows from $A^{(k)}\bar{x} = b^{(k)}$. So $\bar{x}$ is a feasible solution of $LP^{(k)}$ with $c^T M^{(k)} \bar{x} \leq \xi$ because $\bar{x}$ is a nonnegative vector in $H^{(k)}(\xi)$. Then $M^{(k)}\bar{x}$ is feasible for the original problem with (II).

Let ξ' denote the old value of ξ and ξ'' denote the new value computed by the algorithm. We have $c^T M^{(k)} \bar{x} \leq \xi'$. If $\xi'' = UB^{(k)}$, then $\xi'' \geq OPT$ because $UB^{(k)}$ is a valid upper bound

due to the fact that $LP^{(k)}$ has an optimal solution in $[0,1]^n$ whenever the original problem with (II) is feasible; see property (b). If $\xi'' \neq UB^{(k)}$, then

$$\xi' - \xi'' = \frac{\gamma}{n^2}\|P^{(k)}M^{(k)}c\|_1.$$

In this case, using the fact that $\|v\|_1 \leq \sqrt{n}\|v\|$ for all vectors v, we can write

$$\|\pi_{H^{(k)}(\xi')}(\bar{x}) - \pi_{H^{(k)}(\xi'')}(\bar{x})\| \leq \frac{\xi' - \xi''}{\|P^{(k)}M^{(k)}c\|} \leq \frac{\sqrt{n}\gamma}{n^2}.$$

Now we will prove that $\pi_{H^{(k)}(\xi'')}(\bar{x})$ is feasible for $LP^{(k)}$. If $D^{(k)}_{ii} = 0$, then $A^{(k)}x = b^{(k)}$ implies $x_i = 0$. Then $\mathbf{e}_i^T\pi_{H^{(k)}(\xi'')}(\bar{x}) = 0$. Let $D^{(k)}_{ii} = 1$. Then $\mathbf{e}_i^T\pi_{H^{(k)}(\xi')}(\bar{x}) \geq \frac{\gamma}{n}$ because $\bar{x} = \pi_{H^{(k)}(\xi')}(\bar{x})$ due to the fact that $\bar{x} \in H^{(k)}(\xi')$. It follows that

$$\mathbf{e}_i^T\pi_{H^{(k)}(\xi'')}(\bar{x}) \geq \mathbf{e}_i^T\pi_{H^{(k)}(\xi')}(\bar{x}) - \|\pi_{H^{(k)}(\xi')}(\bar{x}) - \pi_{H^{(k)}(\xi'')}(\bar{x})\| \geq \frac{\gamma}{n} - \frac{\sqrt{n}\gamma}{n^2} \geq 0.$$

Thus $\pi_{H^{(k)}(\xi'')}(\bar{x})$ is nonnegative. Since at the same time $\pi_{H^{(k)}(\xi'')}(\bar{x}) \in H^{(k)}(\xi'')$, we have proved that $\pi_{H^{(k)}(\xi'')}(\bar{x})$ is feasible for $LP^{(k)}$. Moreover, $c^TM^{(k)}\pi_{H^{(k)}(\xi'')}(\bar{x}) \leq \xi''$, which implies $\xi'' \geq OPT$ because $M^{(k)}\pi_{H^{(k)}(\xi'')}(\bar{x})$ is feasible for the original problem.

Note that $LP^{(k+1)} = LP^{(k)}$ at any iteration computing a new value of ξ. Then $u^{(k+1)} = u^{(k)}$ satisfies (3.1) because $H^{(k+1)}(\xi'') = H^{(k)}(\xi'') \subseteq H^{(k)}(\xi')$, which implies, for all j with $D^{(k+1)}_{jj} = 1$, and $x \in H^{(k+1)}(\xi'') \cap Q_j$, that

$$\|X^{(k+1)}\mathbf{e}_j - x\|^2 = \|\pi_{H^{(k)}(\xi'')}(X^{(k)}\mathbf{e}_j) - x\|^2 \leq \|X^{(k)}\mathbf{e}_j - x\|^2 \leq u_j^{(k)} = u_j^{(k+1)}.$$

Proof of (a). The property (a) follows from the previous discussion on cases 1 and 2 for step 8 and the end of the above proof of (d). ∎

Theorem 3.1 *Algorithm 3.1 can be implemented to solve a linear optimization problem with (II) in time* $O\left(n^4\sum_{j=1}^{n}\left\lceil\log\frac{2}{\beta_j}\right\rceil\right)$.

Proof. First, let us show that the output is correct if the algorithm terminates. If the algorithm does not decide that the problem is infeasible, then $P^{(k)}M^{(k)}c = \mathbf{0}$ at the last iteration. This equation means that all feasible solutions of $LP^{(k)}$ are at the same time optimal solutions of $LP^{(k)}$. It follows that if $D^{(k)}$ is nonzero, then $x^* = \frac{1}{\sum_{i=1}^{n} D^{(k)}_{ii}}M^{(k)}X^{(k)}D^{(k)}\mathbf{1}$ is an

optimal solution of the original problem because $\bar{x} = \frac{1}{\sum_{i=1}^{n} D_{ii}^{(k)}} X^{(k)} D^{(k)} \mathbf{1}$ is feasible for $LP^{(k)}$ and the optimal value of $LP^{(k)}$ is equal to the optimal value of the original problem, which follows from the property (b) in Lemma 3.1. If $D^{(k)} = O$ and $\mathbf{0}$ is feasible, then $\mathbf{0}$ is an optimal solution because, by Lemma 3.1, $D_{jj}^{(k)} = 0$ implies $x_j^* = 0$ for every optimal solution x^* with (II). If $D^{(k)} = O$ and $\mathbf{0}$ is infeasible, then the original problem with (II) is infeasible. So the output is correct.

Let K denote the number of iterations. We will now prove that K is polynomially bounded.

Note that $u^{(k+1)} \geq \mathbf{0}$ at each iteration. Indeed, $u^{(0)} \geq \mathbf{0}$. Consider iteration k and assume that $u^{(k)} \geq \mathbf{0}$. If a new value ξ is computed, then $u^{(k+1)} = u^{(k)}$. If $u^{(k+1)}$ is computed at step 8, then $u_j < 0$ implies $u_j^{(k+1)} = n$ because $M_{jj}^{(k+1)} \neq M_{jj}^{(k)}$ and $D_{jj}^{(k)} = 1$ in this case. ($u_j < 0$ implies $D_{jj}^{(k)} = 1$ because for all j with $D_{jj}^{(k)} = 0$ we have $\Delta \mathbf{e}_j = \mathbf{0}$ and therefore $u_j = u_j^{(k)} \geq 0$.) It follows that $u^{(k+1)}$ is nonnegative.

All iterations of the algorithm can be split into two classes: those which perform the inner steps of the while-loop and those which compute ξ. Denote the first class by F. It is a subset of $\{0, \ldots, K\}$. That is, $\{0, \ldots, K\} \setminus F$ is the set of iterations that compute ξ.

Consider an iteration $k \in F$. Note that

$$\mathbf{1}^T u^{(k+1)} \leq \mathbf{1}^T u^{(k)} + 5n l_k, \tag{3.3}$$

where $l_k = |\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}|$, which immediately follows from the formula at step 8 because the number of components $u_j^{(k+1)} = n$ set to n is not greater than l_k. (If the respective set is empty, then $l_k = 0$.) Steps 3 and 4 and the fact that the initial matrix $M^{(0)}$ is the identity matrix imply that

$$\sum_{k \in F} l_k \leq \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil .$$

It follows that the number of iterations k in F with $M^{(k+1)} \neq M^{(k)}$ is bounded by $\sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil$. If $M^{(k+1)} = M^{(k)}$, then $(A^{(k+1)}|b^{(k+1)}) = (A^{(k)}|b^{(k)})$. Otherwise, $(A^{(k+1)}|b^{(k+1)})$ can be computed in $O(n^3)$ time. Thus, all the computations of $(A^{(k+1)}|b^{(k+1)})$ in the course

of the algorithm require a running time of

$$O\left(n^3 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right).$$

The same estimate is true for the overall running time for computing $X^{(k+1)}$ and $P^{(k+1)}$ at all iterations with $M^{(k+1)} \neq M^{(k)}$. (Recall that the matrix $P^{(k+1)}$ is necessary for the computation of projections).

Let $k \in F$ and $M^{(k+1)} = M^{(k)}$. Consider $a^T = \mathbf{e}_i^T X^{(k)} D^{(k)}$ for i with $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{1} \leq \gamma$ and $D_{ii}^{(k)} = 1$. We have $a^T \mathbf{1} \leq \gamma$. Note that

$$\left(\max\left(a, \frac{1}{2}\mathbf{e}_i\right) - a\right)^T = \mathbf{e}_i^T \Delta.$$

Lemma 2.2, applied to a defined above and $\delta = \gamma$, implies that

$$\frac{1}{2} - \gamma \leq \mathbf{e}_i^T \Delta \mathbf{1} \leq \|\mathbf{e}_i^T \Delta\| \sqrt{n}. \tag{3.4}$$

For i such that there exists j with $\mathbf{e}_i^T X^{(k)} D^{(k)} \mathbf{e}_j \leq -1$, we have $\mathbf{e}_i^T \Delta \mathbf{e}_j \geq 1$. Thus, summarizing, we obtain

$$\mathbf{e}_i^T (\Delta \circ \Delta) \mathbf{1} = \|\mathbf{e}_i^T \Delta\|^2 \geq \frac{1}{16n}, \quad \forall i \in T,$$

where T is the set defined at iteration k (step 1 of the inner loop). It follows that

$$\mathbf{1}^T u^{(k+1)} \leq \mathbf{1}^T u^{(k)} - \frac{s_k}{16n}, \tag{3.5}$$

where $s_k = |T|$. In order to compute $X^{(k+1)}$, we observe that, since we consider the case $M^{(k+1)} = M^{(k)}$,

$$X^{(k+1)}\mathbf{e}_j = \begin{cases} X^{(k)}\mathbf{e}_j + P^{(k+1)}\Delta\mathbf{e}_j - \dfrac{\max(0, c^T M^{(k+1)}(X^{(k)} + P^{(k+1)}\Delta)\mathbf{e}_j - \xi)}{\|P^{(k+1)}M^{(k+1)}c\|^2} P^{(k+1)}M^{(k+1)}c, \\ X^{(k)}\mathbf{e}_j + P^{(k+1)}\Delta\mathbf{e}_j, \end{cases} \tag{3.6}$$

for j with $D_{jj}^{(k+1)} = 1$. (If $P^{(k+1)}M^{(k+1)}c = \mathbf{0}$, then $X^{(k+1)}\mathbf{e}_j = X^{(k)}\mathbf{e}_j + P^{(k+1)}\Delta\mathbf{e}_j$.) The vector $P^{(k+1)}M^{(k+1)}c$ can be computed in $O(n^2)$ time. The matrix $P^{(k+1)}\Delta$ can be computed in $O(s_k n^2)$ time because all the rows of Δ with $i \notin T$ are zero. It follows that the computation of $X^{(k+1)}$ requires $O(s_k n^2)$ time.

Since $u^{(k)} \geq \mathbf{0}$ for all k,

$$\mathbf{1}^T u^{(K)} \geq 0.$$

Together with this inequality, (3.3) and (3.5) imply that

$$\sum_{k \in F: M^{(k+1)} = M^{(k)}} \frac{s_k}{16n} \leq \mathbf{1}^T u^{(0)} + 5n \sum_{k \in F: M^{(k+1)} \neq M^{(k)}} l_k \leq n^2 + 5n \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil .$$

This implies the overall running time of $O\left(n^4 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$ for computing $X^{(k+1)}$ at all iterations with $M^{(k+1)} = M^{(k)}$. Since $s_k \geq 1$, the above inequality implies that $|F|$ is bounded by $O\left(n^2 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$.

It should be noted that the computations of $(A^{(k+1)}|b^{(k+1)})$ and $X^{(k+1)}$ are computationally the most expensive steps. Each of the other steps requires at most $O(n^2)$ time.

The value ξ is computed at most $O\left(n^2 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$ times in the course of the algorithm. This follows from the fact that $UB^{(k)} - LB^{(k)} = 2\|P^{(k)} M^{(k)} c\|_1$ and ξ is improved by at least $\frac{\gamma}{n^2} \|P^{(k)} M^{(k)} c\|_1$ whenever the algorithm computes a new value of ξ and this new value is not $UB^{(k)}$. Then the number of improvements of ξ is bounded by $8n^2 + 1$ between any two iterations k_1 and k_2, $k_1 < k_2$, such that $M^{(k_1)} = M^{(k_1+1)} = \ldots = M^{(k_2)}$. That is, $|\{0, \ldots, K\} \setminus F| \leq (8n^2 + 1) \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil$. Each iteration k in $\{0, \ldots, K\} \setminus F$ requires $O(n^2)$ time to compute all $X^{(k+1)} \mathbf{e}_j = \pi_{H^{(k)}(\xi)}(X^{(k)} \mathbf{e}_j)$ with $D_{jj}^{(k)} = 1$ for the new ξ because $\pi_{H^{(k)}(\xi')}(X^{(k)} \mathbf{e}_j) = X^{(k)} \mathbf{e}_j$, is already available for the old value ξ' of ξ. Thus the overall running time of iterations k in $\{0, \ldots, K\} \setminus F$ is $O\left(n^4 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$.

Summarizing, we come to the running time $O\left(n^4 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$ of our algorithm. We also conclude that the algorithm requires $O\left(n^2 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$ iterations. ∎

To make the algorithm polynomial in the usual sense, we need only a minor modification as proposed in the lemma below.

Further, we use $\lfloor \cdot \rfloor$ also for componentwise rounding. The projection π_{Q_j} used in the following lemma is computed as $\pi_{Q_j}(v) = \min(\max(v, \frac{1}{2}\mathbf{e}_j), \mathbf{1})$ for each vector v.

Lemma 3.2 *Let $\varepsilon = \frac{1}{3 \cdot 32 \cdot n^3}$. If steps 7 and 8 are replaced by*

$$X^{(k+1)}\mathbf{e}_j := \begin{cases} \pi_{H^{(k+1)}(\xi)}\left(\varepsilon \left\lfloor \frac{1}{\varepsilon}\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j)\right\rfloor\right), & \text{if } (M_{jj}^{(k+1)} = M_{jj}^{(k)}) \wedge (D_{jj}^{(k+1)} = 1) \wedge \\ & \left((M^{(k+1)} \neq M^{(k)}) \vee (\Delta \not\leq \mathbf{1}\mathbf{1}^T)\right), \\ \pi_{H^{(k+1)}(\xi)}\left((X^{(k)} + \varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta\right\rfloor)\mathbf{e}_j\right), & \text{if } (M^{(k+1)} = M^{(k)}) \wedge (\Delta \leq \mathbf{1}\mathbf{1}^T) \\ & \wedge (D_{jj}^{(k+1)} = 1), \\ X^{(k)}\mathbf{e}_j, & D_{jj}^{(k+1)} = 0, \\ \pi_{H^{(k+1)}(\xi)}(\mathbf{0}), & \text{otherwise.} \end{cases}$$

and

$$u_j^{(k+1)} := \begin{cases} u_j + 4|\{i : M_{ii}^{(k+1)} \neq M_{ii}^{(k)}\}| + \frac{1}{32n^2}, & \text{if } M_{jj}^{(k+1)} = M_{jj}^{(k)}, D_{jj}^{(k+1)} = 1, \\ n, & \text{otherwise.} \end{cases}$$

then the algorithm remains correct and the running time is

$$O\left(n^4 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right),$$

and, in the case when A, b, and c are rational, the sizes of the numbers in the course of the algorithm are bounded by a polynomial in the size of the input and in the values $\left\lceil \log \frac{2}{\beta_j} \right\rceil$.

Proof. To prove that Lemma 3.1 is true for the new version of the algorithm, we require only the following additional observation concerning step 8, case 1, in the proof of Lemma 3.1. Consider x' in $H^{(k+1)}(\xi) \cap Q_j$. (We use the same notation as in the proof of Lemma 3.1, step 8, case 1.) There exists $v \in [-\varepsilon, \varepsilon]^n$ such that

$$\left\| \varepsilon \left\lfloor \frac{1}{\varepsilon}\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j)\right\rfloor - x' \right\| = \left\|\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j) - x' + v\right\|$$

$$\leq \left\|\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j) - x'\right\| + \|v\|.$$

We have

$$\left\|\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j) - x'\right\| \leq \sqrt{n}$$

and

$$\|v\| \leq \sqrt{n}\varepsilon.$$

Taking into account $\varepsilon^2 \le \varepsilon$, we can write

$$\left\| \varepsilon \left\lfloor \frac{1}{\varepsilon} \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) \right\rfloor - x' \right\|^2 \le \left\| \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) - x' \right\|^2 + 3n\varepsilon.$$

Moreover,

$$\left\| \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) - x' \right\| \le \left\| (X^{(k)} + \Delta)\mathbf{e}_j - x' \right\|.$$

The analysis of step 8 in the proof of Lemma 3.1 implies

$$\left\| (X^{(k)} + \Delta)\mathbf{e}_j - x' \right\|^2 \le u_j + 4|\{i : M_{ii}^{(k+1)} \ne M_{ii}^{(k)}\}|.$$

Note that

$$\left\| \pi_{H^{(k+1)}(\xi)} \left(\varepsilon \left\lfloor \frac{1}{\varepsilon} \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) \right\rfloor \right) - x' \right\|^2 \le \left\| \varepsilon \left\lfloor \frac{1}{\varepsilon} \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) \right\rfloor - x' \right\|^2.$$

It follows that

$$\left\| \pi_{H^{(k+1)}(\xi)} \left(\varepsilon \left\lfloor \frac{1}{\varepsilon} \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) \right\rfloor \right) - x' \right\|^2 \le u_j + 4|\{i : M_{ii}^{(k+1)} \ne M_{ii}^{(k)}\}| + \frac{1}{32n^2}.$$

Now let $M^{(k+1)} = M^{(k)}$ and $\Delta \le \mathbf{1}\mathbf{1}^T$. The matrix $\varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor$ is nonnegative and $\varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor \le \Delta$. Therefore, it satisfies 2.2, if we write $\varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor$ in place of Δ. Then we have

$$\left\| \left(X^{(k)} + \varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor \right) \mathbf{e}_j - x' \right\|^2 \le \left\| X^{(k)}\mathbf{e}_j - x' \right\|^2 - \left\| \varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor \mathbf{e}_j \right\|^2 \le \left\| X^{(k)}\mathbf{e}_j - x' \right\|^2 - \|\Delta\mathbf{e}_j\|^2 + 2n\varepsilon.$$

It follows that

$$\left\| \pi_{H^{(k+1)}(\xi)} \left(\left(X^{(k)} + \varepsilon \left\lfloor \frac{1}{\varepsilon}\Delta \right\rfloor \right) \mathbf{e}_j \right) - x' \right\|^2 \le u_j + \frac{1}{32n^2}.$$

Following the proof of Lemma 3.1, with the above additional observations for step 8, we conclude that Lemma 3.1 remains correct for the new version of the algorithm.

To obtain the estimate of the running time, we use the proof of Theorem 3.1. We simply replace (3.3) by

$$\mathbf{1}^T u^{(k+1)} \le \mathbf{1}^T u^{(k)} + 5nl_k + \frac{1}{32n},$$

and (3.5) by

$$\mathbf{1}^T u^{(k+1)} \le \mathbf{1}^T u^{(k)} - \frac{s_k}{16n} + \frac{1}{32n} \le \mathbf{1}^T u^{(k)} - \frac{s_k}{32n}.$$

Let F_1 be a subset of iterations of $F_0 = \{k \in F : M^{(k+1)} = M^{(k)}\}$ with $\Delta \not\leq \mathbf{1}\mathbf{1}^T$ and F_2 be a subset of F_0 with $\Delta \leq \mathbf{1}\mathbf{1}^T$.

For all k in F_1,

$$\mathbf{1}^T u^{(k+1)} \leq \mathbf{1}^T u^{(k)} - 1 + \frac{1}{32n}.$$

Then

$$(1 - \frac{1}{32n})|F_1| + \sum_{k \in F_2} \frac{s_k}{32n} \leq \mathbf{1}^T u^{(0)} + 5n \sum_{k \in F : M^{(k+1)} \neq M^{(k)}} l_k \leq n^2 + 5n \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil.$$

The vector $\varepsilon \left\lfloor \frac{1}{\varepsilon} \pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j) \right\rfloor$ can be computed in time $O(n \log n)$ because

$$\|\pi_{Q_j}((X^{(k)} + \Delta)\mathbf{e}_j)\|_\infty \leq 1.$$

So $X^{(k+1)}$ can be computed in time $O(n^3)$ at each iteration k in F_1 and at each iteration k in F with $M^{(k+1)} \neq M^{(k)}$.

The matrix $\Delta^{\#}$ has s_k nonzero rows and can be computed in $O(s_k n \log n)$ time at each iteration k in F_2 because then $\Delta \leq \mathbf{1}\mathbf{1}^T$. The same observation as in the proof of Theorem implies that the complexity of iteration k in F_2 is $O(s_k n^2)$.

Thus, each iteration in F_1 requires $O(n^3)$ time and iteration k in F_2 requires $O(s_k n^2)$ time. The above inequality implies that the number of iterations in F_1 is bounded by $O\left(n \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$. So the overall running time of iterations in F_1 is $O\left(n^4 \sum_{j=1}^{n} \left\lceil \log \frac{2}{\beta_j} \right\rceil\right)$. This estimate is true also for the overall running time of iterations in F_2. Following the proof of Theorem 3.1 in other respects, we obtain the same theoretical estimate of the number of iterations.

In the rest of the proof, we say that a value is polynomially bounded if and only if it is bounded by a polynomial in the size of the input and in the values $\left\lceil \log \frac{2}{\beta_j} \right\rceil$.

It remains to prove that if A, b, and c are rational, then the sizes of the numbers are polynomially bounded in the course of the algorithm. Here we will use the easily provable fact (for example, see [15]) that

$$\text{size}(r_1 + \ldots + r_q) \leq 2(\text{size}(r_1) + \ldots + \text{size}(r_q))$$

for any rational numbers $r_1, \ldots, r_q$. Here, size denotes the binary size. Let K denote the number of iterations performed by the new version of the algorithm. Since K is polynomially bounded, size($\|P^{(k)}M^{(k)}c\|_1$) is polynomially bounded at each iteration. The size of $UB^{(k)}$ is polynomially bounded as well. It follows from the above property of the sizes of rational numbers that the size of ξ is polynomially bounded in the course of the algorithm. Then the size of $\pi_{H^{(k+1)}(\xi)}\left(\varepsilon\left\lfloor\frac{1}{\varepsilon}\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j)\right\rfloor\right)$ is polynomially bounded because the size of $\varepsilon\left\lfloor\frac{1}{\varepsilon}\pi_{Q_j}((X^{(k)}+\Delta)\mathbf{e}_j)\right\rfloor$ is bounded by a polynomial in n. It follows that the size of $X^{(k)}$ is polynomially bounded for all k in F_1 and for all k in F with $M^{(k+1)} \neq M^{(k)}$.

Consider an iteration k in F_2 and an index j with $D_{jj}^{(k+1)} = 1$. Denote $\Delta^{\#} = \varepsilon\left\lfloor\frac{1}{\varepsilon}\Delta\right\rfloor$. Without loss of generality, we choose the case

$$X^{(k+1)}\mathbf{e}_j = X^{(k)}\mathbf{e}_j + P^{(k+1)}\Delta^{\#}\mathbf{e}_j - \frac{\max(0, c^T M^{(k+1)}(X^{(k)} + P^{(k+1)}\Delta^{\#})\mathbf{e}_j - \xi)}{\|P^{(k+1)}M^{(k+1)}c\|^2}P^{(k+1)}M^{(k+1)}c.$$

Consider iteration t such that

$$t = \min\{l \in F_2 : l \leq k\}.$$

Taking into account that $M^{(k+1)} = M^{(k)}$ and the components of $\Delta^{\#}\mathbf{e}_j$ belong to $\left\{0, \frac{1}{8n}, \frac{2}{8n}, \ldots, 1\right\}$ because $k \in F_2$, and multiplying both sides of the above equation by $c^T M^{(k+1)}$ (note that $P^{(k+1)}$ is idempotent, which means $c^T M^{(k+1)}P^{(k+1)}M^{(k+1)}c = \|P^{(k+1)}M^{(k+1)}c\|^2$) we can write

$$c^T M^{(k+1)}X^{(k+1)}\mathbf{e}_j = \begin{cases} \xi \\ c^T M^{(k)}(X^{(k)} + P^{(k)}\Delta^{\#})\mathbf{e}_j \end{cases} = \begin{cases} \xi \\ \hat{\xi} + \sum_{l=r}^{k+1} c^T M^{(l)}P^{(l)}v^{(l)} \\ c^T M^{(t)}X^{(t)}\mathbf{e}_j + \sum_{l=t}^{k+1} c^T M^{(l)}P^{(l)}w^{(l)} \end{cases},$$

where $\hat{\xi}$ is the value of ξ at some iteration $r \in \{t, \ldots, k-1\}$ and $v^{(l)}, w^{(l)} \in \{0, \frac{1}{8n}, \frac{2}{8n}, \ldots, 1\}^n$. The size of $X^{(t)}\mathbf{e}_j = \pi_{H^{(t)}(\xi')}(\mathbf{0})$ is polynomially bounded. It follows that the binary size of $c^T M^{(k+1)}X^{(k+1)}$ is polynomially bounded because k is bounded by K and the sizes of $c^T M^{(t)}P^{(t)}\mathbf{e}_j$, $c^T M^{(l)}P^{(l)}v^{(l)}$, and $c^T M^{(l)}P^{(l)}w^{(l)}$ are polynomially bounded.

The remaining cases with $M_{jj}^{(k+1)} \neq M_{jj}^{(k)}$ and $D_{jj}^{(k+1)} = 0$ are obvious. If $M_{jj}^{(k+1)} \neq M_{jj}^{(k)}$ and $D_{jj}^{(k+1)} = 1$, then $X^{(k+1)}\mathbf{e}_j = \pi_{H^{(k+1)}(\xi)}(\mathbf{0})$. The size of $\pi_{H^{(k+1)}(\xi)}(\mathbf{0})$ is polynomially

bounded because ξ and $M^{(k+1)}$ are of polynomial size. If $D_{jj}^{(k+1)} = 0$, then $X^{(k+1)}\mathbf{e}_j = X^{(k)}\mathbf{e}_j$ and we can prove by induction that the size of $X^{(k+1)}\mathbf{e}_j$ is polynomially bounded.

Thus we have proved that the size of

$$\frac{\max(0, c^T M^{(k+1)}(X^{(k)} + P^{(k+1)}\Delta^{\#})\mathbf{e}_j - \xi)}{\|P^{(k+1)}M^{(k+1)}c\|^2}$$

is polynomially bounded for any iteration k in F. For all iterations $k \notin F$, i.e., for those which compute new values of ξ, we have $X^{(k+1)}\mathbf{e}_j = X^{(k)}\mathbf{e}_j$.

It follows that $X^{(k+1)}\mathbf{e}_j$ can be expressed as a sum of a polynomially bounded number of vectors of polynomial size. The property of rational numbers mentioned above implies that $X^{(k+1)}\mathbf{e}_j$ is of polynomial size.

Thus, the size of $X^{(k)}$ is polynomially bounded at any iteration. Summarizing, we conclude that the sizes of all of the other numbers computed in the course of the algorithm are polynomially bounded.

■

Now we obtain the main result of this paper.

Theorem 3.2 *Algorithm 3.1 modified as proposed in Lemma 3.2 is a polynomial algorithm for linear programming. Moreover, it is a strongly polynomial algorithm for linear optimization problems with property (I).*

Proof. It is well known that linear optimization problems are polynomially reducible to linear feasibility problems, i.e., to systems of linear inequalities, with lower and upper bounds on variables; see Schrijver [22]. So without loss of generality we assume that the problem in question is either bounded or infeasible. Then, in polynomial time, we can find α and β, both of polynomial size, such that if the problem is feasible then there exists an optimal solution x^* with $x_j^* \in \{0\} \cup [\alpha, \beta]$ for all j. Multiplying each component of c and each column of A by β, we obtain a problem with (II) for $\beta_j = \alpha/\beta$. Now we apply Algorithm 3.1. If it returns an optimal solution, we multiply each component of this solution by β to obtain an optimal solution of the original problem.

In the case of a problem with (I), we replace c^T by c^T $\mathrm{diag}(\alpha_1 2^{p(n)}, \ldots, \alpha_n 2^{p(n)})$ and A by

A diag$(\alpha_1 2^{p(n)}, \ldots, \alpha_n 2^{p(n)})$. This leads to (II) for $\beta_j = 1/2^{p(n)}$. Then we apply the modified Algorithm 3.1 and obtain a running time of $O(n^5 p(n))$. ∎

Consider an integer linear problem

$$\min\{c^T x : Ax = b, \mathbf{0} \leq x \leq v, x \in \mathbb{Z}^n\},$$

where v is an integer vector. By the LP relaxation of this problem we understand the linear problem obtained by omitting the integrality constraints. Usually, when solving an LP relaxation, we are interested in a feasible solution whose objective value is a lower bound on the optimal value of the respective integer problem. The following corollary states that there is a polynomial algorithm, for finding such a solution, whose running time depends only on n and size(v).

Corollary 3.1 *There exists an algorithm which either proves that the integer linear problem is infeasible or finds a feasible solution x^* of the LP relaxation such that $c^T x^*$ is a lower bound on the optimal value of the integer linear problem in $O(n^4 size(v))$ time.*

Proof. Let $\bar{A} = A$ diag(v) and $\bar{c} =$ diag$(v)c$. Consider

$$\min\{\bar{c}^T \bar{x} : \bar{A}\bar{x} = b, \bar{x} + z = 1, \bar{x} \geq \mathbf{0}, z \geq \mathbf{0}\}.$$

Run Algorithm 3.1 under the assumption that there exists an optimal solution with $\bar{x}_j$ and z_j in $\{0\} \cup [1/v_j, 1]$. The algorithm requires $O(n^4 \text{size}(v))$ time. (Note that we do not assert that such a solution really exists.) If the algorithm does not compute a new value of ξ at least once, then the integer linear problem is infeasible or $\mathbf{0}$ is an optimal solution if $\mathbf{0}$ is feasible. If ξ is computed at least once, let ξ^* be the minimum value of ξ computed in the course of the algorithm. From the feasible solution of the respective problem $LP^{(k)}$, found by the algorithm when computing ξ^*, we can obtain a feasible solution $(\bar{x}^*; z^*)$ of the above linear problem with $c^T \text{diag}(v)\bar{x}^* = \xi^*$. From the proof of Lemma 3.1 it follows that ξ^* is a lower bound on the objective values of solutions with $\bar{x}_j$ in $\{0\} \cup [1/v_j, 1]$ and z_j in $\{0\} \cup [1/v_j, 1]$. At the same time ξ^* is a lower bound on the optimal value of the integer linear problem. We have $c^T x^* = \xi^*$, where $x^* = \text{diag}(v)\bar{x}^*$ is a feasible solution of the LP relaxation of the integer linear problem. ∎

So the algorithm given in the proof of the above corollary finds a lower bound which is not less than that delivered by the LP relaxation. The running time of this algorithm depends only on the dimension and the binary size of v. If size(v) is bounded by a polynomial in n, then the running time is strongly polynomial. In the case of a 0-1 integer problem, the complexity of the algorithm is $O(n^5)$. The modified ellipsoid method developed in Section 2.2, under the assumption that it applies to optimization problems, would require $O(n^6 \log n)$ time. The special structure of ellipsoids used by the modified ellipsoid method suggests that it can be possible to reduce the complexity of each iteration to $O(n^2)$ by means of low-rank updates. If such an improvement is possible, then we would get the running time $O(n^5 \log n)$ for the modified ellipsoid method, in the context of solving LP relaxations of 0-1 integer problems.

Now consider a more general nonlinear problem

$$\min\{c^T x : Ax = b, x \in S_1 \times \ldots \times S_n\},$$

where $S_j \subseteq \{0\} \cup [v_j, \infty)$ for each j and v is a real vector. Consider an LP relaxation of the form

$$\min\{c^T x : Ax = b, x \geq \mathbf{0}\}.$$

Corollary 3.2 *Given a real vector w with $w \geq v$, we can either prove that the nonlinear problem has no solutions in $\{x : \mathbf{0} \leq x \leq w\}$ or find a feasible solution x^* of the above LP relaxation such that $c^T x^*$ is a lower bound on the optimal value of the nonlinear problem in $O\left(n^4 \sum_{j=1}^{n} \log \left\lceil \frac{2w_j}{v_j} \right\rceil\right)$ time.*

Proof Let $\bar{A} = A \operatorname{diag}(w)$ and $\bar{c} = \operatorname{diag}(w)c$. Consider

$$\min\{\bar{c}^T \bar{x} : \bar{A}\bar{x} = b, \bar{x} \geq \mathbf{0}\}.$$

Run Algorithm 3.1 under the assumption that there exists an optimal solution with $\bar{x}_j$ in $\{0\} \cup [v_j/w_j, 1]$. The algorithm runs in $O\left(n^4 \sum_{j=1}^{n} \log \left\lceil \frac{2w_j}{v_j} \right\rceil\right)$ time. If the algorithm does not compute a new value of ξ at least once, then the nonlinear problem has no feasible solutions with $\mathbf{0} \leq x \leq w$ or $\mathbf{0}$ is an optimal solution if $\mathbf{0}$ is feasible. If ξ is computed at least

once, let ξ^* be the minimum value of ξ computed in the course of the algorithm. From the feasible solution of the respective problem $LP^{(k)}$, found by the algorithm when computing ξ^*, we can obtain a feasible solution $\bar{x}^*$, of the above linear problem, with $c^T \mathrm{diag}(w)\bar{x}^* = \xi^*$. From the proof of Lemma 3.1 it follows that ξ^* is a lower bound on the objective values of solutions $\bar{x}$ with $\bar{x}_j$ in $\{0\} \cup [v_j/w_j, 1]$. At the same time, if the nonlinear problem has an optimal solution in $\{x : \mathbf{0} \leq x \leq w\}$, then ξ^* is a lower bound on the optimal value of the nonlinear problem. We have $c^T x^* = \xi^*$, where $x^* = \mathrm{diag}(w)\bar{x}^*$ is a feasible solution of the LP relaxation of the integer linear problem. ∎

If the values $\log \left\lceil \frac{2w_j}{v_j} \right\rceil$ are bounded by a polynomial in the number of variables, then we obtain a strongly polynomial running time for the above LP relaxation.

4 Conclusions

We considered a new polynomial algorithm for linear programming based on a general method, proposed in Section 2.5. The algorithm is strongly polynomial if it is known that there exists an optimal solution where each component is either zero or lies in an interval whose endpoints satisfy certain conditions. As a corollary, we obtained some new complexity results related to finding lower bounds on optimal values of some nonlinear problems.

References

[1] Agmon, Sh. 1954. The relaxation method for linear inequalities. *Canad. J. Math.* **6**, 382-392.

[2] Basu, A., Junod, M. and De Loera, J. 2014. On Chubanov's Method for Linear Programming. *INFORMS Journal on Computing* **26**, 336-350.

[3] Blum, L., Shub, M., and Smale, S. 1989. On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bulletin (New Series) of the American Mathematical Society* **21**.

[4] Boyd, S. and Vandenberghe, L. Convex Optimization. Cambridge University Press, 2003.

[5] Cheney, W., and Goldstein, A. 1959. Proximity maps for convex sets. *Proceedings of the AMS* **10**, 448-450.

[6] Chubanov, S. 2012. A strongly polynomial algorithm for linear systems having a binary solution. *Mathematical Programming* **134**, 533-570.

[7] Chubanov, S. 2014. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*. DOI 10.1007/s10107-014-0823-8

[8] Dantzig, G. B. 1955. Optimal solutions of a dynamic Leontief model with substitution, Econometrica 23 295-302.

[9] Dattorro, J. 2005. Convex Optimization & Euclidean Distance Geometry, Meboo publishing, USA.

[10] Edmonds, J. 1967. Systems of distinct representatives and linear algebra. *J. Res. Nat. Bur. Standards* **71 B**, 241-245.

[11] Gill, P., Murray, W., Saunders, M., Tomlin, J., and Wright, M. 1986. On projected Newton barrier methods for linear programming and an equivalence to Karmarkar's projective method. *Mathematical Programming* **36**, p. 183-209.

[12] Goffin, J.L. 1982. On the non-polynomiality of the relaxation method for systems of linear inequalities. *Mathematical Programming* **22**, 93-103.

[13] Karmarkar, N. 1984. A new polynomial-time algorithm for linear programming. *Combinatorica* **4**, 353-395.

[14] Khachiyan, L.G. 1979. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR 244* (English translation: Soviet Math. Dokl. **20**, 191-194).

[15] Korte, B., and Vygen, J. 2002. Combinatorial optimization: theory and algorithms. Springer.

[16] Maurras, J.-F., Truemper, K., and Akgül, M. 1981. Polynomial algorithms for a class of linear programs. *Mathematical Programming* **21**, 121-136.

[17] Megiddo, N. 1983. Towards a genuinely polynomial algorithm for linear programming. *SIAM J. Comput.* **12**, 347-353.

[18] Motzkin, Th., Schoenberg, I. J. 1954. The relaxation method for linear inequalities. *Canad. J. Math.* **6**, 393-404.

[19] Nemirovski. A.S. 1988. A new polynomial algorithm for linear programming (Russian). *Dokl. Akad. Nauk SSSR* **298**(6), 1321-1325. Translation in *Soviet Math. Dokl.* **37**(1), 264-269, 1988.

[20] Renegar, J. 1988. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming* **40**, 59-93.

[21] Roos, C. 2014. On Chubanov's method for solving a homogeneous inequality system. *Optimization Online*.

[22] Schrijver, A. 1998. Theory of linear and integer programming.

[23] Smale, S. 1998. Mathematical Problems for the Next Century. *Mathematical Intelligencer* **20**, 7-15.

[24] Tardos, E. 1986. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research* **34**, 250-256.

[25] Vavasis, S., and Ye, Y. 1996. A primal-dual interior-point method whose running time depends only on the constraint matrix. *Mathematical Programming* **74**, 79-120.

[26] Végh, L.A., Zambelli, G. 2014. A polynomial projection-type algorithm for linear programming. *Operations Research Letters* **42**, 91-96.

[27] Ye, Y. 2005. A new complexity result on solving the Markov Decision Problem. *Mathematics of Operations Research* **30**, 733-749.

[28] Ye, Y. 2011. The simplex and policy-iteration methods are strongly polynomial for the Markov Decision Problem with a fixed discount rate. *Mathematics of Operations Research* **36**, 593 - 603.