A New Algorithm for Linear Programming in Critical Systems

Adrien Chan-Hon-Tong

A new algorithm for linear programming in critical systems

Adrien CHAN-HON-TONG
ONERA Université Paris Saclay, 91140, France

January 16, 2023

Abstract

In critical systems (e.g. for core airplane functions), codes should both never fail, in particular they should be robust to numerical instabilities, and, they should reuse certified routines.

Yet, the combination of this two constraints is often an issue. For example, the interior-point algorithms for linear programming have higher complexity than expected when requiring simultaneously numerical robustness and no custom routines.

Instead, this paper presents a new algorithm, which has good time complexity even with a naive implementation.

Foreword

This paper is an extended version of conference paper [5]. The main difference with [5] is that the proof is significantly simplified: there is no need to have a distinct part for each of the two so-called phases of the Newton descent anymore. Also, this paper describes, more precisely than [5], the interest of the new method to deal with critical systems regarding the state-of-the-art methods.

Notations

This paper takes advantage of standard matrix notation. In particular, if u, v are 2 vectors i.e. $u, v \in \mathbb{R}^N$ $u^T v$ is the scalar product of the two vectors, because those vectors are seen like matrices in $\mathbb{R}^{N\times 1}$, thus, u^T is a matrix in $\mathbb{R}^{1\times N}$, and thus, the matrix product $u^T \times v$ (written $u^T v$) is $\sum_n u_n v_n$.

Also, the scalar product of a vector u with itself (i.e. $u^T u$) will be written $||u||_2^2$ in this paper using standard notation for the L_2 norm.

1 Introduction

1.1 Linear programming

Linear programming is a central optimization problem which aims at producing either a solution x or a certificate that the following problem is infeasible or

unbounded:

$$\min_{x\in\mathbb{Q}^N,\ Ax\geq b} c^T x \tag{1}$$

where $A \in \mathbb{Z}^{M\times N}$ is a matrix and $b \in \mathbb{Z}^M$, $c \in \mathbb{Z}^N$ are two vectors with M being the number of constraints/rows of A and N the number of variables/columns of A. Assuming $N = O(M)$, the state-of-the-art is interior-point algorithms (e.g.[17, 20, 23]) which solve linear programs with total binary size L in (at most) $\widetilde{O}(\sqrt{M}L)$ Newton steps[1].

Using standard linear algebra routines, one can perform each of those Newton step (mainly a matrix inversion) in $\widetilde{O}(M^\omega)$ arithmetic operations where ω is the coefficient of the matrix multiplication (known to be equivalent to the one of the matrix inversion [19] i.e. 3 with simple algorithm but 2.38 with [2]). Recently, [23] (a deterministic version of [7]) proves that a specific data structure allows to save a $\sqrt{M}$ factor from those specific matrix inversions.

Linear programming is equivalent to strict linear feasibility which aims at solving

$$\text{find } x \in X_A = \{\chi \in \mathbb{Q}^N,\ A\chi > \mathbf{0}\} \text{ assuming } X_A \neq \emptyset \tag{2}$$

where $A \in \mathbb{Z}^{M\times N}$ and $\mathbf{0}$ is the vector full of 0. The problem is sometimes presented as finding x such that $Ax \geq \mathbf{1}$ (where $\mathbf{1}$ is the vector full of 1) to avoid strict inequalities but all those three formulations are equivalent.

The algorithm presented in section 2 will deal with this strict linear feasibility for convenience without restricting the generality as a pre-processing and a post-processing allow to deal with general linear programming without higher time complexity (This is presented in appendix for completeness but this conversion is not a contribution).

1.2 Software in critical systems

Independently from linear programming, this paper focuses on codes performing critical tasks. In such context, the primary target of algorithm implementation is usually the ability to be certified and/or to reuse (as much as possible) standard routines, rather than improving the practical speed of the solver.

1.2.1 Binary vs arithmetic operations

The first issue in this context is numerical stability and/or binary considerations. Indeed, in classical implementations, operations are realized with constant precision e.g. IEEE 754 floating point convention. But, such implementations may have numerical failure: for example if a value is 2^{-32767}, the IEEE 754 floating point convention will round it to 0, while, 0 may not be correct. Of course, with strong assumption on the inputs, it is sometimes possible to rely on such a floating point representation safely (e.g. [4]). But, without any assumption,

[1] ($\widetilde{O}(.)$ notation will be used instead of $O(.)$ to express the fact that log factors are omitted e.g. $O(M\log(M)) = \widetilde{O}(M)$).

there is many algorithm where avoiding numerical issue can not be done with such constant representation.

Yet, one solution of this problem is *arbitrarily large integer representation*: the computation on $\mathbb{Q}$ can be arbitrarily precise by representing integers as unbounded sequence of digits and rational as fraction of such integers (other representations may be possible like interval representation [22] but they are out of the scope of this paper).

However, under arbitrarily large integer representation, considering that arithmetic operations are *single operation* (e.g. that matrix inversion can be done in $\widetilde{O}(M^\omega)$ operations) is only half of the story. Indeed, with such representation, one has to take into account the binary size of intermediate numbers which can grow during the algorithm. For example, Gaussian elimination is a M^3 algorithm when considering arithmetic operations, and, it can be carefully implemented in $O(M^3)$. However, if implemented naively, then Gaussian elimination becomes exponential [10].

1.2.2 Binary consideration in linear programming

Linear programming does not allow to avoid the previous discussion about binary issues. On one hand, [11, 8] manage to apply formal verification to the ellipsoid algorithm [15] with simple floating point representation using prior on the inputs. This result is important to advances on the application of a linear program solver on a critical task. For example, DO-178 requires that codes in core airplane system should be produced using strict code production process, and, they could ideally be verified using formal tools. But, this result does not hold without strong assumptions on the inputs: in worst cases, numbers encountered in linear program solvers can be large.

On the other hand, most linear programming algorithms can be adapted for arbitrarily large integer representation. But, this comes at the cost of a higher time complexity for [15, 17] and/or the requirement for custom linear algebra routine for [20] and/or some custom data-structure for [23].

1.3 Contribution: dealing with binary issue with standard routines

The previous problem of dealing with binary considerations is just one of all the requirements for implementation on critical platform. Another one is to rely on standard routines and/or with the simplest possible algorithm. This explains why there is still research on old algorithm like the ellipsoid method [11, 8]: even if it is the worst polynomial time algorithm for linear programming, it has some good features like the fact that it does not require matrix inversion. This also explains why, neither [20] nor [23] are fully satisfying as they require either custom routines or custom data-structure.

In the same way, this paper introduces a new algorithm for linear programming in critical context called self-concordant Perceptron with the following features:

- It can be implemented naively with standard linear algebra routines without damage.
- It performs at most $O(ML)$ steps (i.e. $\sqrt{M}$ more than [20] but much less than the ellipsoid method).
- It deals with strict linear feasibility, but, it is significantly different than [9, 18]. Indeed, it deals with strict linear feasibility is a way close to the original Perceptron [21].
- It relies on self-concordance theory. But, it relies on the so-called *first phase of Newton descent* (while [17, 20] rely on the *second phase* and on the notion of central-path).

The presented algorithm is not a breakthrough for time complexity theory seeing [20, 23]. Also, it is incremental as it relies on the self-concordance theory [16]. But, this algorithm should interest safety community and more generally a broader audience as:

- It represents an interesting contribution for critical systems as it performs correctly even under naive implementation.
- It is a non trivial application of the self-concordance theory on a problem taking advantage of strict linear feasibility formulation.
- It is significantly different from [20, 23]. For example, strong negative results from [1] do not directly apply to it.

The algorithm is precisely described in section 2 (with the proof of convergence) after a more detailed presentation of the existing issues with the current state-of-the-art.

1.4 Issues with current state-of-the-art

1.4.1 Ellipsoid method

The ellipsoid method [15, 12] relies on an internal representation x, E initialized as $\mathbf{0}, 2^L\text{Diag}(\mathbf{1})$ (where $\text{Diag}(u)$ with u a vector is the diagonal square matrix with same size as the vector whose value i, i is u_i) and performs M^2L steps

1. **find** k, $A_k x - b_k < 0$
2. $x = x - \frac{1}{M+1}\frac{EA_k^T}{\sqrt{A_k E A_k^T}}$
3. $E = \frac{M^2}{(M+1)^2}(E - \frac{2}{M+1}\frac{A_k E^T E A_k^T}{A_k E A_k^T})$

In the original paper, computations are claimed to be safe when numerical error is lower than $\exp(-10ML)$. This explains why the complexity of this algorithm is given as $O(M^6L^2)$ [14] while arithmetic complexity is only $O(M^4L)$ [15].

1.4.2 Classical log barrier

The classical log barrier algorithm [17, 3] minimizes the function $G(x,\mu) = c^T x - \mu \sum_m \log(A_m x - b_m)$ by performing Newton descent on x and halving μ:

1. $x = x - \frac{1}{\sqrt{(\nabla_x G)^T (\nabla_x^2 G)^{-1} (\nabla_x G)}} (\nabla_x^2 G)^{-1} (\nabla_x G)$
2. $\mu = \frac{\mu}{2}$

However, $\mu = \frac{\mu}{2}$ is clearly an unacceptable setting from binary point of view as the number of steps is $\sqrt{ML}$ in worst cases leading to $\mu^* = \frac{1}{2^{\sqrt{ML}}} \mu_{start}$. Currently, it is not clear that there are instances requiring such number of steps, yet [1] recently proves that with very large L, there exists instances for which more that 2^M steps are required. Anyway, this implies that numbers should be represented with at least $O(\sqrt{ML})$ digits leading to some extra factor into complexity.

1.4.3 Path following

The path following algorithm [20] has the same arithmetic complexity than [17], but, avoids the issue of the small μ by relying on an updating depending on M on a slightly different function $G(x,\mu) = -\sqrt{M} \log(c^T x - \mu) - \sum_m \log(A_m x - b_m)$:

1. $\mu = (1 - \frac{1}{\sqrt{M}})\mu + \frac{1}{\sqrt{M}}(c^T x)$
2. $x = x - \frac{1}{\sqrt{(\nabla_x G)^T (\nabla_x^2 G)^{-1} (\nabla_x G)}} (\nabla_x^2 G)^{-1} (\nabla_x G)$

This way, [20] proves that linear programming admits a $O(M^\omega \sqrt{ML^2})$ binary time complexity with arbitrarily large integer representation.

Yet, if one considers the hessian related to $-\sum_m \log(A_m x - b_m)$ which will be written H in the following. Then, $H = A^T \text{Diag}(Ax - b)^{-2} A$ or

$$\forall i,j \; H_{i,j} = \sum_m \frac{A_{m,i} A_{m,j}}{(A_m x - b_m)^2}$$

and, one could remark that this matrix can have coefficient as small as $2^{-O(ML)}$ leading to large numbers when computing the inverse with standard routines.

Currently, the eigen values of this matrix are within $[2^{-O(L)}, 2^{O(L)}]$ making it possible to perform inversion with rounding (as pointed by [20]). But, this may require custom matrix inversion routines.

1.5 Overview

To finish this overview of the state-of-the-art, the Chubanov algorithm [6] contains an halving operation $A_k^T = \frac{1}{2} \times A_k^T$ (like the μ of the classical log barrier) which is a binary issue as the number of steps is ML. The situation is the same for the *Rescaling Phase* of [18]. Finally, [23] has the best time complexity[2]

[2]It also performs $O(\sqrt{ML})$ steps, yet, the average complexity of each step is better.

nowadays, but it structurally requires a very specific data structure.

Instead, this paper introduces a new algorithm related to interior point algorithms but with some differences which make the algorithm more stable under a naive implementation. This discussion is summarized in table 1.

Algorithm	Complexity	Issues for critical implementation
Ellipsoid method ([15])	$\widetilde{O}(M^4L)$	ML precision
Deterministic rescaling ([18])	$\widetilde{O}(M^3\sqrt{M}L)$	ML halving
Chubanov method ([6])	$\widetilde{O}(M^\omega ML)$	ML halving
Classical log barrier ([17])	$\widetilde{O}(M^\omega\sqrt{M}L)$	$\sqrt{M}L$ halving
Path following ([20])	$\widetilde{O}(M^\omega\sqrt{M}L)$	custom matrix inversion
[20] with data structure ([23])	$\widetilde{\mathbf{O}}(\mathbf{M}^\omega\mathbf{L})$	custom data-structure
this paper	$\widetilde{O}(M^\omega ML)$	**none of the above**

Table 1: Issues which can be encountered for different algorithms from the state-of-the-art when implementing them on a critical plateform: most issues are related to binary complexity but it can be about the requirement for custom matrix inversion routines (while relying of standard routines could be prefered).

2 The self-concordant Perceptron

The algorithm introduced by this paper is called the self-concordant Perceptron (as it is a smoothed version of the original Perceptron). It is presented in table 2. The algorithm is a Newton descent on

$$F_A(v) = \frac{1}{2}v^TAA^Tv - \sum_{m\in\{1,\dots,M\}} \log(v_m) \tag{3}$$

starting from $\frac{1}{\Upsilon_A}\mathbf{1}$ with $\Upsilon_A = \sqrt{\max_m A_m A_m^T}$. An additional 1D operation $v \leftarrow \sqrt{\frac{M}{v^TAA^Tv}}v$ and, a ceiling is performed at each step.

To be completely exhaustive, operations like $\sqrt{v^TAA^Tv}$ are not possible on $\mathbb{Q}$ (the major relevance of this algorithm is to tackle binary issues). Fortunately, the function being convex, trivial 2-approximations will be sufficient to ensure convergence:

- $v^TAA^Tv = M$ is replaced by $v^TAA^Tv \leq 4M$
- Computing $\lambda_{F_A}^2 = (\nabla_v F_A)^T(\nabla_v^2 F_A)^{-1}(\nabla_v F_A)$ is possible, and, thus computing a 2 approximation of λ_{F_A} is trivial (currently, λ_{F_A} is lower than 1 in practice, so just considering the damped update $v \leftarrow v - \frac{1}{2}(\nabla_v^2 F_A)^{-1}(\nabla_v F_A)$ is almost always sufficient).

Index $_A$ for F, Υ, Γ will be omitted when not ambiguous.

2.1 Pre-requite of the proof

self-concordant theory: The proof of the central theorem of self-concordant theory presented bellow can be found in [16].
If $\Psi(x)$ is a self-concordant function (mainly any sum of quadratic, linear, constant and $-\log$ term), with a minimum Ψ^*, then, the Newton descent starting from x_{start} allows to compute x_ϵ such as $\Psi(x_\epsilon) - \Psi^* \leq \varepsilon$ in $\widetilde{O}(\Psi(x_{start}) - \Psi^* + \log\log(\frac{1}{\varepsilon}))$ damped Newton steps. Each step is:

- $\lambda_\Psi(x) \leftarrow \sqrt{(\nabla_x \Psi)^T (\nabla_x^2 \Psi)^{-1} (\nabla_x \Psi)}$
- $x \leftarrow x - \frac{1}{1+\lambda_\Psi(x)} (\nabla_x^2 \Psi)^{-1} (\nabla_x \Psi)$

Importantly, this descent has 2 so-called phases:

- While $\lambda_\Psi(x) \geq \frac{1}{4}$, each damped Newton step decreases Ψ of at least $\frac{1}{4} - \log(\frac{5}{4}) \geq \frac{1}{50}$. This so-called first phase can not last more than $50 \times (\Psi(x_{start}) - \Psi^*)$ damped Newton steps.
- As soon as one has computed any x_{phase} with $\lambda_\Psi(x_{phase}) \leq \frac{1}{4}$, then, only $\widetilde{O}(\log\log(\frac{1}{\varepsilon}))$ additional steps are required to get x_ϵ such as $\Psi(x_\epsilon) - \Psi^* \leq \varepsilon$. This is the so-called second phase with quadratic convergence (i.e. $\log\log(\varepsilon)$ steps lead to a precision ϵ). Importantly, $\lambda_\Psi(x_{phase}) \leq \frac{1}{4} \Rightarrow \Psi(x_{phase}) - \Psi^* \leq \frac{1}{4}$.

2.2 An overview of the key mechanisms of the self-concordant Perceptron

Self_concordant_Perceptron(A)
  F being symbolically $\frac{1}{2} v^T A A^T v - \sum_m \log(v_m)$
  $\Upsilon \leftarrow \sqrt{\max_m A_m A_m^T}$
  $v \leftarrow \frac{1}{\Upsilon}\mathbf{1}$; $\Gamma \leftarrow 1000 M \sqrt{M} \Upsilon$
  while $\neg(AA^T v > \mathbf{0})$ **do**
    $v \leftarrow v - \frac{1}{1+\lambda_F(v)} (\nabla_v^2 F)^{-1} (\nabla_v F)$
    if $\lambda_F(v) \geq \frac{1}{4}$ **then**
      $v \leftarrow \sqrt{\frac{M}{v^T A A^T v}} v$
      $v \leftarrow \frac{1}{\Gamma} \times \text{int}(\Gamma \times v + 1)$
    end if
  end while
  return v

Table 2: self-concordant Perceptron algorithm.

2.2.1 Ideas behind the Newton descent:

Considering the self-concordant algorithm from table 2, the main mechanism behind the proof is that the Newton descent applied to the function F will always decrease F by at least $1/50$ (as the optimization takes place in the so-called first phase see 2.1).

Indeed, the Newton descent allows to go from a function with multiple variables $F(x)$ to a 1D function $\rho(t) = F(x - t(\nabla_v^2 F)^{-1}(\nabla_v F))$. Now, $\rho(t) = \alpha\frac{t^2}{2} + \beta t - \sum_m \log(a_m t + 1)$ by definition of F, and, $\rho'(t) = -\rho''(t) = -\lambda^2$ as a property of the Newton direction.

Then, by applying Taylor extension to ρ, one has $\forall t \geq 0,\ \exists \tau \in [0,t],\ \rho(t) = t\rho'(0) + \frac{t^2}{2}\rho''(\tau) = -t\lambda^2 + \frac{t^2}{2}(\alpha + \sum_m \frac{a_m^2}{(a_m\tau+1)^2})$. As, $\rho''(0) = \lambda^2$ implies that $|a_m| < \lambda$, it comes that $\rho(t) \leq -t\lambda^2 + \frac{t^2}{2}(\alpha + \sum_m \frac{a_m^2}{(1-\lambda\tau)^2}) \leq -t\lambda^2 + \frac{t^2}{2}\frac{a+\sum_m a_m^2}{(1-\lambda t)^2} \leq -t\lambda^2 + \frac{t^2}{2}\frac{\lambda^2}{(1-\lambda t)^2}$.

Then, independently from the matrix A, a lower bound of the improvement only based on λ can be found as $\rho(t) \leq -t\lambda^2 + \frac{t^2}{2}\frac{\lambda^2}{(1-\lambda t)^2}$ which is higher than $1/4$ (otherwise the optimization enters the so-called second phase see 2.1). This way, an absolute-constant-and-simple decrease for each Newton step is obtained. This is the core of the proof: with such property, it is possible to design a simple strategy to deal with binary properties without requiring specific routines to perform the Newton step.

The other important point is that computing the inverse of $AA^T + Diag(v)^{-2}$ can be simpler than the one of $A^T Diag(Ax-b)^{-2}A$. Indeed, in the first case, multiplying by $Diag(v)^2$ allows to restore an integer matrix, while, it is required to multiply by the product of M numerators from $Ax - b$ in the second case.

Those ideas are formally proven bellow.

2.2.2 Structure of the proof:

First, a set of lemmas will allow to find the correct values to apply the self-concordant theory to function F_A. This will lead to the theorem 1.

Yet, without rounding, naive Newton descent has an exponential binary complexity like naive Gaussian elimination. But, theorem 2 will provide a bound on the effect of a ceiling on $F(v)$.

By combining the arithmetic property of Newton descent, and the bound on the ceiling, the theorem 3 will state the complexity of the self-concordant Perceptron. Precisely,

- Lemma 4 requires lemma 3 which requires lemma 2.
- Lemmas 1 and 5 are independent from those ones.
- Then, lemma 6 requires lemma 5.
- Then, theorem 1 requires lemmas 1, 4 and 6.

- Theorem 2 is independent.
- Finally, theorem 3 combines theorems 1 and 2 (and lemma 5).

2.3 Proof

Lemma 1:
$\forall A \in \mathbb{Q}^{M\times N}$, $x \in \mathbb{Q}^N$ such that $Ax \geq \mathbf{1}$, and, $v \geq \mathbf{0}$, then, $\frac{||v||_2^2}{||x||_2^2} \leq ||A^T v||_2^2$, and, thus, $||A^T v||_2^2 \leq 4M \Rightarrow v \leq 2\sqrt{M}||x||_2 \times \mathbf{1}$

Proof. Cauchy inequality applied to $x^T(A^T v)$ gives: $x^T(A^T v) \leq ||x||_2 \times ||A^T v||_2$.

But, $x^T(A^T v) = (Ax)^T v \geq \mathbf{1}^T v$ as $v \geq \mathbf{0}$ and $Ax \geq \mathbf{1}$. Thus, $\mathbf{1}^T v \leq ||x||_2 \times ||A^T v||_2$.

As each side is positive, one could take the square (and push $||x||_2$ to the left), this gives $\frac{(\mathbf{1}^T v)^2}{||x||_2^2} \leq ||A^T v||_2^2$. Yet, as $v \geq \mathbf{0}$, $v^T v \leq (\mathbf{1}^T v)^2$.

Second part is just injection of $||A^T v||_2^2 \leq 4M$. □

Lemma 2:
Let $f(t) = \frac{1}{2||x||_2^2} t^2 - \log(t)$ with any vector x with $||x||_2 \geq 1$, then, f is lower bounded with a minimum $f^* = \frac{1-\log(||x||_2)}{2} \geq -\log(||x||_2)$.

Proof. f is a continuous function from $]0, \infty[$ to $\mathbb{R}$. $f(t) \underset{t\to 0}{\to} \infty$ due to the $-\log$, and, $f(t) \underset{t\to\infty}{\to} \infty$ due to the t^2. So, f is lower bounded with a minimum. As f is smooth, this minimum is solution of $f'(t) = \frac{t}{||x||_2^2} - \frac{1}{t} = 0$ i.e. $t^* = ||x||_2$ and $f^* = f(||x||_2)$. □

Importantly, it is assumed in linear feasible that $X_A \neq \emptyset$.
So this assumption will be omitted in all following lemmas/theorems.

Lemma 3:
F_A is lower bounded and $F_A^* \geq -M\log(||x||_2)$ (for any x such that $Ax \geq \mathbf{1}$).

Proof. As $X_A \neq \emptyset$, then, $\exists x, Ax \geq \mathbf{1}$. But, following lemma 1, it holds that $F_A(v) \geq \frac{v^T v}{2x^T x} - \sum_m \log(v_m) = \sum_m f(v_m)$ (with the function f introduced in lemma 2). So, $F_A(v) \geq \sum_m f^* \geq -M\log(||x||_2)$ following lemma 2. Finally, as for all m, $F_A(v) \geq f(v_m) + (M-1)f^*$ and $f(t) \to \infty$ in 0 or ∞, then, it means F can not admit an infimum on the border of $]0,\infty[^M$. So the property of being lower bounded (by Mf^*) without infimum at the border implies that F_A has a minimum F_A^*, and so $F_A^* \geq Mf^*$. □

Lemma 4:
$F_A(v) - F_A^* \leq \min_m \frac{1}{v_m^2 A_m A_m^T + 1} \Rightarrow AA^T v > \mathbf{0}$

Proof. Let assume that there exists k such as $A_k A^T v \leq 0$, and, let introduce $w = v + t\mathbf{1}_k$ i.e. $w_m = v_m$ if $m \neq k$ and $w_k = v_k + t$.

$F_A(w_k) = \frac{1}{2}(v + t\mathbf{1}_k)^T AA^T(v + t\mathbf{1}_k) - \sum_m \log(v_m) + \log(v_k) - \log(v_k + t) = F_A(v) + tA_k A^T v + \frac{1}{2}t^2 A_k A_k^T - \log(v_k + t) + \log(v_k)$. But, $A_k A^T v \leq 0$, so $F_A(w_k) \leq F_A(v) + \frac{1}{2}t^2 A_k A_k^T - \log(v_k + t) + \log(v_k)$, and, it is clear that for $0 \leq t \ll 1$, $F_A(w_k) < F_A(v)$ (because this is $-\log(v_k + t)$ at first order).

Precisely, one could define $\Phi(t) = F_A(v) + \frac{1}{2}t^2 A_k A_k^T - \log(v_k + t) + \log(v_k)$. Then, $\Phi'(t) = A_k A_k^T t - \frac{1}{t+v_k}$ and $\Phi''(t) = A_k A_k^T + \frac{1}{(t+v_k)^2}$ and $\Phi'''(t) = -\frac{2}{(t+v_k)^3}$. As, $\Phi'''(t) \leq 0$ and $t \geq 0$, $\Phi(t) \leq \Phi(0) + t\Phi'(0) + \frac{t^2}{2}\Phi''(0)$ i.e.

$$\Phi(t) \leq -\frac{t}{v_k} + \frac{t^2}{2}(A_k A_k^T + \frac{1}{v_k^2})$$

In particular, for $t = \frac{v_k}{v_k^2 \times A_k A_k^T + 1}$, $F_A(w) \leq F_A(v) - \frac{1}{2}\frac{1}{v_k^2 \times A_k A_k^T + 1}$. But, this is not possible if $F_A(v)$ is closer than F_A^* by this value.

□

Lemma 5:
$F_A(\sqrt{\frac{M}{v^T AA^T v}} \times v) \leq F_A(v)$, and, $F_A(v) - F^* \leq \frac{1}{16} \Rightarrow v^T AA^T v \leq 4M$.

Proof. Considering the function $t \to F_A(t \times v) = \frac{1}{2}v^T AA^T v \times t^2 - \sum_m \log(v_m) - M\log(t)$ trivially proves that $F_A(v)$ decreases when v is normalized such as $v^T AA^T v$ goes closer to M. In particular, if $v^T AA^T v \geq 4M$, then, $v \leftarrow \frac{v}{2}$ allows to decrease F by $3M - M\log(2) \geq \frac{1}{16}$. So, this is not possible if $F(v) - F^*$ is lower than this value.

□

Lemma 6:

$$F_A(v) - F_A^* \leq \min(\frac{1}{4Mx^T x\Upsilon^2 + 1}, \frac{1}{16}) \Rightarrow AA^T v > \mathbf{0}$$

Proof. Lemma 5 proves that $v^T AA^T v \leq M$ (because $F_A(v) - F_A^* \leq \frac{1}{16}$).

Then, this is just lemma 4 combined with $v \leq \sqrt{2}||x||_2\mathbf{1}$ from lemma 1 (as $A_m A_m^T \leq \Gamma^2$ by definition). □

Theorem 1:
Damped Newton descent on F_A starting from any v_{start} will terminate eventually returning v such that $AA^T v > \mathbf{0}$ at least when $F_A(v) - F_A^* \leq \min(\frac{1}{4Mx^T x\Upsilon^2+1}, \frac{1}{16})$.
And this will not require more than $O(F_A(v_{start}) - F^* + \log\log(4Mx^T x\Upsilon^2 + 1))$ Newton steps.
In particular, from $v = \frac{1}{\Upsilon} \times \mathbf{1}$, this will require no more than $\widetilde{O}(ML)$ Newton steps in the so-called first phase, and, only $\widetilde{O}(\log(L))$ in the so-called second phase are required to terminate.

Proof. The first part of this theorem is just the self-concordant theory applied to F_A with $\varepsilon = \min(\frac{1}{4Mx^Tx\Upsilon^2+1}, \frac{1}{16})$. This holds because F_A has a minimum as proven in lemma 3 because $X_A \neq \emptyset$ (this last assumption $X_A \neq \emptyset$ is critical otherwise F can goes to $-\infty$).
Yet, this ε value leads to a solution of the original linear feasibility problem from lemma 6.
The second point is based on the classical results that the maximal norm of a vector defined by a linear system of total binary size L is $O(L)$. Thus, $\log(x^Tx) = \widetilde{O}(L)$ as if $X_A \neq \emptyset$, then, there exists x entirely defined by a submatrix of A in X_A. Then, $F(\frac{1}{\Upsilon}\mathbf{1}) \leq M^2 - M\log(\Upsilon) = O(ML)$ (Cauchy for the quadratic term and definition of L for $\log(\Upsilon) \leq L$), and, $-F^* \leq M\log(x^Tx) = O(ML)$ due to lemma 2. So, the so-called first phase lasts no more than $O(ML)$ steps. Then, the so-called second phase lasts only $O(\log\log(4Mx^Tx\Upsilon^2+1))$ which is just $\log(L)$ steps (definition of L + bound on x + lemma 2). □

Remark: At this point, it is proven that Newton descent on F converges. But Newton descent without dealing with binary size of the variable is exponential like naive Gaussian elimination. Yet, the self-concordant Perceptron is a Newton descent with a simple strategy to deal with variable binary size. This last point is proven in next theorems.

Theorem 2:
Assume that $v^TAA^Tv \leq 4M$, then:

$$\forall \varpi \in \left[0, \frac{1}{\Gamma_A}\right]^M, \qquad F(v+\varpi) \leq F(v) + \frac{1}{200}$$

In particular, $\forall v$,

$$F\begin{pmatrix} \frac{int(\Gamma_A \times v_1)+1}{\Gamma_A} \\ \dots \\ \frac{int(\Gamma_A \times v_M)+1}{\Gamma_A} \end{pmatrix} \leq F(v) + \frac{1}{200}$$

Proof. First, the log part only decreases when adding $\varpi \geq \mathbf{0}$, thus, only the quadratic part should be considered. So $F(v+\varpi) \leq F(v) + \frac{1}{2}\varpi^TAA^T\varpi + \varpi^TAA^Tv$.

But, $A^T\varpi = \sum_m \varpi_m A_m^T$ so $||A^T\varpi|| \leq \sum_m \varpi_m ||A_m^T|| \leq ||\varpi||_\infty M\Upsilon \leq \frac{1}{500\sqrt{M}}$, and $||A^T\varpi||_2^2 = \varpi^TAA^T\varpi \leq \frac{1}{(1000)^2M}$.

So, $\varpi^TAA^Tv \leq \sqrt{\varpi^TAA^T\varpi \times v^TAA^Tv} \leq \sqrt{\frac{1}{(500)^2M} \times 4M} \leq \frac{1}{250}$ (from Cauchy). And, $\frac{1}{2}\varpi^TAA^T\varpi \leq \frac{1}{2\times(1000)^2M} \leq \frac{50}{1000}$. Thus, it holds that $F(v+\varpi) \leq F(v) + \frac{1}{200}$.

Then, $int(t)+1$ is a special case of $t+\tau, \tau \in [0,1]$, so the presented rounding scheme correspond to add $\varpi \in \left[0, \frac{1}{\Gamma_A}\right]^M$. □

Theorem 3:

The self-concordant Perceptron described in table 2 always terminates in less than $\widetilde{O}(ML)$ steps eventually returning v such that $AA^Tv > \mathbf{0}$. During (almost) all the algorithm, all values of v have a common denominator of Γ, and, all numerators are bounded by $2^{O(L)}$.
Finally, this is done by computing the inverse of $\nabla_v^2 F = AA^T + \text{Diag}(v)^{-2}$. For that, this matrix should be scaled to $\mathcal{H} = \Gamma^2(\text{Diag}(v)AA^T\text{Diag}(v) + I)$ to recover integer values before inversion. Yet, $\mathcal{H}$ has the interesting property that $\forall i, j\ \mathcal{H}_{i,j} \leq 2^{O(L)}$ without requiring any kind of rounding.

Proof. The self-concordant Perceptron described in table 2 is a Newton descent upgraded with a scaling and a ceiling. But, the theorem 1 proves that the Newton descent alone converges and decreases F by at least $\frac{1}{50}$ each steps.

Then, the scaling will not increase F (this is lemma 5), but, will ensure $v^TAA^Tv \leq 4M$. Thus, the theorem 2 holds and proves that the ceiling will not increase F by more than $\frac{1}{200}$.

So, each step of self-concordant Perceptron decreases F by $\frac{1}{50} - 0 - \frac{1}{200}$ (effect of a single Newton step - scaling - ceiling) i.e. there is still a constant decrease of F during all the so-called first phase. So it will terminate with only twice number of steps.

So, the self-concordant Perceptron (with mastered binary size) converges with the same number of steps (in big-O) than the Newton descent (which is naively exponential form binary point of view).

This proves the first part.

Then, the second part of the theorem is directly implied by the first (and lemma 6) because that $\mathcal{H} = \Gamma^2(\text{Diag}(v)AA^T\text{Diag}(v) + I)$ is an integer matrix with values bounded by $2^{O(L)}$ (it is true for $\Gamma\text{Diag}(v)$ and A, and, not modified by product as, numerators are bounded by $2\sqrt{Mx^Tx}\Gamma$ which is $\widetilde{O}(2^L)$).

□

remark: During all the algorithm $v^TAA^Tv \leq 4M$. Indeed, during the so-called first phase this is due to the integer scaling of v (this scaling decreases F) to allow an easy rounding. And, during the second phase this holds naturally (without the need of a scaling) and ensures the convergence of the algorithm. So this property is usefull for two different reasons. Also, the so-called second phase is negligible with only $O(\log(L))$ step explaining why ceiling can be deactivated during this phase.

2.4 Conclusion

This paper introduces the self-concordant Perceptron which converges in $O(ML)$ steps, with a common denominator and all numerators of it internal values v requiring only L digits each. In addition, each step is mainly the inversion of $\mathcal{H} = \Gamma^2(\text{Diag}(v)AA^T\text{Diag}(v) + I)$ which is a not singular integer matrix with each coefficient bounded by $2^{O(L)}$, while, the classical constraint matrix $H = A^T\text{Diag}(Ax - b)^{-2}A$ naively requires ML digits per value.

Thus, even if this algorithm does not compete against [23] in general, it can be relevant for critical contexts as it is simpler to implement (using any pre-existing linear algebra routines) with still good times complexity.

2.5 Conflict of Interest

The author declares that he has no conflict of interest.

References

[1] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Log-barrier interior point methods are not strongly polynomial. *Journal on Applied Algebra and Geometry*, 2018.

[2] Andris Ambainis, Yuval Filmus, and François Le Gall. Fast matrix multiplication: limitations of the coppersmith-winograd method. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 585–593, 2015.

[3] Erling D Anderson, Jacek Gondzio, Csaba Mészáros, and Xiaojie Xu. Implementation of interior-point methods for large scale linear programs. In *Interior Point Methods of Mathematical Programming*, pages 189–252. Springer, 1996.

[4] Sylvie Boldo. Floats and ropes: a case study for formal numerical program verification. In *International Colloquium on Automata, Languages, and Programming*, pages 91–102. Springer, 2009.

[5] Adrien Chan-Hon-Tong. Solving linear programming while tackling number representation issues. In *Proceedings of the 11th International Conference on Operations Research and Enterprise Systems - ICORES,*, pages 40–47. INSTICC, SciTePress, 2022.

[6] Sergei Chubanov. A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, 153(2):687–713, 2015.

[7] Michael B Cohen, Yin Tat Lee, and Zhao Song. Solving linear programs in the current matrix multiplication time. In *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, 2019.

[8] Raphael P Cohen. *Formal verification and validation of convex optimization algorithms for model predictive control.* PhD thesis, Georgia Institute of Technology, 2018.

[9] John Dunagan and Santosh Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Mathematical Programming*, 114(1):101–114, 2008.

[10] Xin Gui Fang and George Havas. On the worst-case complexity of integer gaussian elimination. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 28–31, 1997.

[11] Eric M Feron, Raphael P Cohen, Guillaume Davy, and Pierre-Loic Garoche. Validation of convex optimization algorithms and credible implementation for model predictive control. In *AIAA Information Systems-AIAA Infotech@ Aerospace*, page 0562. 2017.

[12] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1981.

[13] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Fundamentals of convex analysis*. Springer Science & Business Media, 2004.

[14] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984.

[15] Leonid Khachiyan. A polynomial algorithm for linear programming. *Doklady Akademii Nauk SSSR*, 1979.

[16] Arkadi Nemirovski. Interior point polynomial time methods in convex programming. *Lecture notes*, 42(16):3215–3224, 2004.

[17] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*. Siam, 1994.

[18] Javier Peña and Negar Soheili. A deterministic rescaled perceptron algorithm. *Mathematical Programming*, 155(1-2):497–510, 2016.

[19] Marko D. Petković and Predrag S. Stanimirović. Generalized matrix inversion is not harder than matrix multiplication. *Journal of Computational and Applied Mathematics*, 230(1):270–282, 2009.

[20] James Renegar. A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical programming*, 40(1):59–93, 1988.

[21] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[22] Herry Suprajitno and I Bin Mohd. Linear programming with interval arithmetic. *Int. J. Contemp. Math. Sciences*, 5(7):323–332, 2010.

[23] Jan van den Brand. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 259–278. SIAM, 2020.

APPENDIX

Reduction to linear feasibility

This paper deals with strict linear feasibility instead of linear programming for convenience but both are known to be equivalent. This appendix details the conversion between them. But this is definitely not a contribution of the paper.

Let $\min_{x, Ax \geq b} c^T x$ be a linear program with total binary size L and M constraints, $N = O(M)$ variables, but, without any assumption: it can be unbounded and/or infeasible and c can be $\mathbf{0}$ (when it is just a problem of satisfying constraints).

Primal dual

First, primal dual theorem [13] proves that: $\min_{\chi, A\chi \geq b} c^T \chi$ is bounded and feasible if and only if there exist x, y such as:

$$Ax \geq b, c^T x - b^T y \geq 0, b^T y - c^T x \geq 0, A^T y \geq c^T, -A^T y \geq -c^T, -y \geq \mathbf{0}$$

and, in this case x is the optimal solution of the input linear program.

So, one is able to either solve the input linear program or finding a certificate, by solving $\mathcal{A}_1 \chi \geq \beta_1$ with $\beta_1^T = (b^T \quad 0 \quad 0 \quad c^T \quad -c^T \quad \mathbf{0}^T)$, and,

$$\mathcal{A}_1 = \begin{pmatrix} A & \mathbf{0} \\ c^T & -b^T \\ -c^T & b^T \\ \mathbf{0} & A^T \\ \mathbf{0} & -A^T \\ \mathbf{0} & -Diag(\mathbf{1}) \end{pmatrix}$$

Let stress that the total binary size and the shape of $\mathcal{A}_1$ are equivalent to the ones of A.

Ensuring a solution

This new problem $\mathcal{A}_1 \chi \geq \beta_1$ will provide either a solution to the original one or it is not feasible and this is a certificate.

Now, as being sure to deal with a bounded-and-feasible problem can be convenient, one could form: $\min_{\chi, \tau, \mathcal{A}_1 \chi + \tau \geq \beta_1, \tau \geq 0} \tau$ which is bounded ($\tau \geq 0$) and always feasible (a trivial initialization is $\chi = \mathbf{0}$ and $\tau \gg 1$).

Indeed, solving this bounded-and-feasible problem will give a solution of $\mathcal{A}_1 \chi \geq \beta_1$ (if $\tau^* = 0$) or a certificate (if $\tau^* > 0$).

Let stress that the total binary size and the shape of this problem (just an additional variable and constraint) are still equivalent to the ones of A.

Primal dual again

Then, one can form the primal dual written $\mathcal{A}_2 \chi \geq \beta_2$ of this last problem $\min_{\chi, \tau, \mathcal{A}_1 \chi + \tau \geq \beta_1, \tau \geq 0} \tau$.

The interest of this second primal dual is that is will always be feasible because the new primal (which is the first primal dual) is feasible and bounded. So, any admissible point of this always-feasible-second-primal-dual will give a solution of the first one.

Then, one can form $\min_{\chi,\tau,\mathcal{A}_2\chi+\tau\geq\beta_2,\tau\geq 0}\tau$ by adding again a constraint and a variable to the second primal dual.

Why someone should do that ? Because this last problem is feasible, bounded and the optimal solution is known to be 0. And, again it is possible to recover solution from the original problem while the total binary size and the shape are still equivalent to the initial ones.

Jumping to strict linear inequality set

So, one has computed a new problem $\min_{\chi,\tau,\mathcal{A}_2\chi+\tau\geq\beta_2,\tau\geq 0}\tau$ equivalent to the original instance but which is always feasible, bounded and with optimal value 0.

Now, the key idea is to consider the set of strict linear inequality constraints $\mathcal{A}_2\chi+\tau > \beta_2, \tau > 0, -2^{O(L)}\times\tau > -1$ with the $2^{O(L)}$ being higher than $Det(\mathcal{A}_2)+1$ written as $\mathcal{A}_3\chi > \beta_3$.

This last problem has necessarily a solution because $\min_{\chi,\tau,\mathcal{A}_2\chi+\tau\geq\beta_2,\tau\geq 0}\tau$ is feasible with optimal solution 0, i.e. there exists x^*,τ^* with $\tau^*=0$. So, just considering $x=x^*$ and $\tau=2^{-(O(L)+1)}$ gives a solution to the set of strict linear inequality constraints.

Also, the total binary size and the shape are still equivalent to the initial ones. Indeed, a $2^{O(L)}$ coefficient is added, but, the binary size of this coefficient is just $O(L)$ so the total binary size just goes from $\widetilde{O}(L)$ to $\widetilde{O}(L)+O(L)=\widetilde{O}(L)$.

Post-processing any solution of the strict linear inequality constraint $\mathcal{A}_3\chi > \beta_3$ to recover the ones of $\min_{\chi,\tau,\mathcal{A}_2\chi+\tau\geq\beta_2,\tau\geq 0}\tau$ is tackled after the final transformation.

Enforcing homogeneity

At this point, one wants to solve a problem $\mathcal{A}_3\chi > \beta_3$ which is directly equivalent to $\mathcal{A}_3 x-\beta_3 t > \mathbf{0}, t > 0$. Indeed, let consider x,t such that $\mathcal{A}_3 x-\beta_3 t > \mathbf{0}, t > 0$ a fortiori $t\neq 0$ and $t\geq 0$, so $\mathcal{A}_3\frac{x}{t}-\beta_3\frac{t}{t} > \mathbf{0}$ i.e. $\mathcal{A}_3\frac{x}{t} > \beta_3$. Inversely, if χ is a solution of the first, $x=\chi$, $\tau=1$ is a solution of the second (and again, the sizes are equivalent).

Purifying solution of strict linear feasibility

The only missing step in the pipeline is how one can retrieve the optimal solution (known to have $\tau=0$) of $\min_{\chi,\tau,\mathcal{A}_2\chi+\tau\geq\beta_2,\tau\geq 0}\tau$ from a solution of $\mathcal{A}_2\chi+\tau > \beta_2, \tau > 0, -2^{O(L)}\times\tau > -1$.

Let consider χ,τ such as $\mathcal{A}_2\chi+\tau > \beta_2, \tau > 0, -2^{O(L)}\times\tau > -1$. Let $I=\{i, \mathcal{A}_2\chi+\tau=\beta_2\}$, $J=\{j,\chi_j=0\}$.

If, there is $\omega, \mathcal{A}_{2,I}\omega = \mathbf{1}, \omega_J = \mathbf{0}$, then, one can update $\chi = \chi + t\omega, \tau = \tau - t$ while maintaining $\mathcal{A}_{2,I}\chi + \tau = \beta_2$, $\chi_J = \mathbf{0}$ and also $\mathcal{A}_2\chi + \tau \geq \beta_2$ until t increases such that a new constraint enters into I or a new component becomes 0 (i.e. a component enters in J).

But, I, J increase as sets, so this process can not last more than $2M$ steps. At the end, $\mathcal{A}_2\chi + \tau \geq \beta_2$, and the system $\mathcal{A}_{2,I}\chi + \tau = \beta_2$, $\chi_J = \mathbf{0}$ define an unique χ, τ i.e. χ, τ are defined by the linear system $\mathcal{A}_{2,I}\chi = \beta_2$, $\chi_J = \mathbf{0}$.

In particular, τ can be written as a fraction of determinant extracted from $\mathcal{A}_2$ (due to Cramer rules) i.e. either $\tau = 0$ or $\tau \geq \frac{1}{Det(\mathcal{A}_2)}$. But, this last option is impossible, because, τ has decreased during the purification, and thus, it verifies $-2^{O(L)} \times \tau > -1$ i.e. $\tau \leq \frac{1}{2^{O(L)}} \leq \frac{1}{Det(\mathcal{A}_2)}$.

So, in this particular case (because, it starts from a point very close to the optimum), the greedy purification leads to a solution with at most M matrix inversions i.e. $MM^{\omega}L$ binary operations (this is a matrix extracted from A so this is the correct binary complexity).

Importantly, let stress that in $\mathcal{A}_2$ there is no additional term in $2^{O(L)}$ which appears only in $\mathcal{A}_3$. So, the purification only consider a submatrix extracted from $\mathcal{A}_2$ and not from $\mathcal{A}_3$ so there is no issue with a determinant which will become larger due to the additional term $2^{O(L)}$.

Finally the complete process first builds a linear program with good assumption (feasible, bounded, with known optimal value of 0) thanks to 2 primal dual steps, then, it builds a strict linear feasibility instance (knowing that greedy purification of any solution of this last problem will allow to recover a solution of the original linear program). As pointed the number of variables and constraints is not scaled by more than 16 and the total binary size not scaled by more than 4. So, strict linear feasibility (with assumption of a solution) is correctly equivalent to linear programming.

Final overview

A summary of the discussion of the equivalence of *linear programming* and *strict linear feasibility* is presented with pseudo-code. Assume $\mathbf{algo}_1(A)$ takes A and returns one x such as $Ax > \mathbf{0}$ if one exists, then:

$\mathbf{algo}_2(A, b)$

$xt \leftarrow \mathbf{algo}_1\left(\begin{pmatrix} A & -b \\ 0 & 1 \end{pmatrix}\right)$

$\mathbf{return}\ (xt\_n)_{n\in\{1,\ldots,N\}}/xt_{N+1}$

takes A, b and returns one x with $Ax > b$ if one exists.

$\mathbf{algo}_3(A, b)$

$\Gamma \leftarrow$ Hadamard bound on A

$xt = \mathbf{algo}_2\left(\begin{pmatrix} A & \mathbf{1} \\ 0 & t \\ 0 & -\Gamma \end{pmatrix}, \begin{pmatrix} b \\ 0 \\ -1 \end{pmatrix}\right)$

$x_2, t_2 \leftarrow (xt\_n)_{n\in\{1,\ldots,N\}}/xt_{N+1}$

$S \leftarrow \{m, A_m x_2 + t_2 = b_m\}$

while $\exists \chi, A_S \chi = \mathbf{1}$ **do**
$x_2 \leftarrow x_2 + \lambda\chi$, $t_2 \leftarrow x_2 - \lambda$
with λ maximal such that $Ax_2 + t_2\mathbf{1} \geq b$
$S \leftarrow \{m, A_m x_2 + t_2 = b_m\}$
end while
return x_2

takes A, b and returns one x with $Ax \geq b$ if one exists.

$\mathbf{algo}_4(A, b)$
$A_p \leftarrow \begin{pmatrix} A & \mathbf{1} \\ 0 & 1 \end{pmatrix}$, $b_p \leftarrow \begin{pmatrix} b \\ 0 \end{pmatrix}$, $c_p \leftarrow (\mathbf{0} \quad 1)$
compute $A_{dual}, b_{dual}, c_{dual}$ with duality theory
$\chi \leftarrow \mathbf{algo}_3 \left(\begin{pmatrix} A_p & \mathbf{0} \\ \mathbf{0} & A_{dual} \\ c_p & -c_{dual} \\ -c_p & c_{dual} \end{pmatrix}, \begin{pmatrix} b_p \\ b_{dual} \\ 0 \\ 0 \end{pmatrix} \right)$
$x \leftarrow (\chi_m)_{m \in \{1,\dots,M\}}$, $t \leftarrow \chi_{M+1}$
return x, t

takes A, b returns one x, t such that $t > 0$ means that there is no $Ax \geq b$, and, $t = 0$ means that $Ax \geq b$.

$\mathbf{algo}_5(A, b, c)$
compute $A_{dual}, b_{dual}, c_{dual}$ with duality theory
$x, t \leftarrow \mathbf{algo}_4(A, b)$
$y, \tau \leftarrow \mathbf{algo}_4(A_{dual}, b_{dual})$
if $t > 0$ or $\tau > 0$ **then**
return infeasible ($t > 0$) or unbounded ($\tau > 0$)
else

$$\chi \leftarrow \mathbf{algo}_3 \left(\begin{pmatrix} A & \mathbf{0} \\ \mathbf{0} & A_{dual} \\ c & -c_{dual} \\ -c & c_{dual} \end{pmatrix}, \begin{pmatrix} b \\ b_{dual} \\ 0 \\ 0 \end{pmatrix} \right)$$

return $(\chi_m)_{m \in \{1,\dots,M\}}$
end if

is a standard linear programming solver.