Binary complexity in linear programming.

Adrien Chan-Hon-Tong

Binary complexity in linear programming.

Adrien CHAN-HON-TONG

February 17, 2022

Abstract

Naive Gaussian elimination is known to be exponential when done with arbitrary large integer representation. This highlights that binary properties of algorithms are (at least theoretically) important. Yet, state of the art of linear programming only weakly tackles this issue. Indeed, this paper stresses some difficulties which can arise when solving linear programming with classical methods under arbitrary large integer representation setting. Then, it introduces a new polynomial times algorithm for linear programming with better binary properties.

1 Introduction

1.1 Linear programming

Linear programming is a central optimization problem which aims to produces 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 $b \in \mathbb{Z}^M$, $c \in \mathbb{Z}^N$ 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 central-path log-barrier [12] and/or path-following [15] algorithms which solves linear programs with total binary size L in less than $\widetilde{O}(\sqrt{M}L)$ Newton steps. As each Newton step is mainly the resolution of an $M \times M$ linear system, the arithmetic time complexity of those algorithms is $\widetilde{O}(M^\omega \sqrt{M}L)$ ($\widetilde{O}(.)$ notation will be used instead of $O(.)$ to express the fact that log factors are omitted) where ω is the coefficient of matrix inversion (known to be equivalent to the one of matrix inversion [14]) i.e. 3 with simple algorithm but 2.38 with [1]. Faster randomized algorithms like [5] are not in the scope of this paper.

As it is important for the following of this paper, the duality theorem allows to prove that, from theoretical point of view, linear programming is equivalent to 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 $Ax \geq \mathbf{1}$ (with $\mathbf{1}$ is the vector full of 1) to avoid strict inequalities, yet, those formulations are equivalent.

This linear feasibility problem comes usually behind a pre processing because not all matrix A ensure that $X_A \neq \emptyset$. However, given any linear programming problem with size $M, N = O(M)$ and total binary size L (it could also be $Ax \geq \mathbf{1}$ without assumption that there is a solution), a trivial pre processing can compute a matrix $\mathcal{A} \in \mathbb{Z}^{O(M)\times O(N)}$ with total binary size $O(L)$ (no more than $16M$ constraints and $4L$ are required) such that $X_{\mathcal{A}} \neq \emptyset$, and, such that any $x \in X_{\mathcal{A}}$ can be trivially post processed to recover either a solution of the initial problem or a certificate that it is infeasible/unbounded (pre and post processing being both trivial and strongly polynomial).

1.2 Binary vs arithmetic operations

Now, considering that arithmetic operations are *single operation* and/or that matrix inversion can be done in $\widetilde{O}(M^\omega)$ operations is only half of the story. It is true by considering operation on $\mathbb{Z}$ (or $\mathbb{Q}$) as 1 operation. This assumption matches the practice where operations are realized with fixed precision (for example with classical floating point representation). But, this is not correct from theoretical point of view, because a fixed size precision open the door to numerical instabilities. Typically, if the solution of a problem is 2^{-L} with $L > 32766$, then, the floating point approximation will be 0 with classical IEEE 754 floating point convention. But, 0 may not be a solution.

Thus, from theoretical point of view, it is required to have arbitrary large representation i.e. in the simplest form, numbers are integers coded as unbounded sequence of digits, or fraction of such integers. But, this structural protection against numerical instability comes at cost that arithmetic operation and binary one do not match anymore: it depends on binary sizes of intermediate numbers which can growth during the algorithm. Typically, naive Gaussian elimination is exponential [7] when using arbitrary large integer representation (while it is a M^3 algorithm when considering arithmetic operations).

Hopefully, there is careful implementations of Gaussian elimination which do not suffer from those binary issues. Yet, it should be pointed (because it will be important for the following of this paper) that:

- Multiplying 2 integer matrix with the absolute value of each entry being bounded by $O(2^B)$ can be done in $O(M^\omega B)$ binary operations because binary size does not increase during matrix multiplication (*absolute value of each entry* will be shorten as *value* in the paper).
- Similarly, multiplying 2 integer matrix with total binary size $O(L)$ can also be done in $O(M^\omega L)$ binary operations because a fortiori each value is bounded by $O(2^L)$ (even if it may be a lose bound when $L = M^2 B$).
- Also, solving $Ax = b$ with an integer matrix A with total binary size L and an integer vector b (with total binary size lower than L) can still be done in $O(M^\omega L)$ binary operations because Cramer rule + Hadarmar bound ensure that each value in the inverse matrix is no more than $O(2^L)$ and that a common denominator (being $Det(A)$) lower than $O(2^L)$ can be found.
- But, if the assumption is that each value of A, b is bounded by $O(2^B)$, then, the situation is different because $Det(A)$ is not bounded by 2^B but by 2^{MB}, resulting in a binary complexity of $O(M^\omega MB)$ for an

exact computation (even relying on efficient Cholesky decomposition: the extra factor M can not be avoided as it appears in the output).

1.3 Binary issue in linear programming

This paper is about binary issues, as the ones pointed in [7], in context of linear programming. Indeed, most classical methods have binary issues (maybe not as critical as in [7]) but potentially affecting their binary complexities.

1.3.1 Ellipsoid method

The ellipsoid method [10, 8] 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, computation are claimed to be safe when rounding error is lower than $\exp(-10ML)$. So, basically, a denominator $O(\frac{1}{2^{ML}})$ is required leading to extra factor between arithmetic and binary complexity because number should thus be encoded on ML digits and not just L. This explains why complexity of this algorithm is sometimes given as $O(M^6L^2)$ while arithmetic complexity is only $O(M^4L)$.

1.3.2 Log barrier

The log barrier algorithm [11, 12] 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. $\mu = \frac{\mu}{2}$

This algorithm only requires $\sqrt{ML}$ steps with each step being basically a matrix inversion, leading to the best known arithmetic complexity of $\widetilde{O}(M^\omega \times \sqrt{ML})$.

However, $\mu = \frac{\mu}{2}$ is clearly an unacceptable setting from binary point of view as the number of step is $\sqrt{ML}$: it leads to $\mu^* = \frac{1}{2^{\sqrt{ML}}}\mu_{start}$. In this condition, there is no way to represent number with less than $O(\sqrt{ML})$ digits. Thus, again, there is necessarily extra factor between binary and arithmetic complexity.

1.3.3 Path following (and Karmarkar algorithm)

Path following algorithm [15] (which is in some way the final version of Karmarkar algorithm [9]) offers the same arithmetic complexity from log barrier, but, avoids the issue of the halving 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, $\mu^* = 2^{O(L)}\mu_{start}$ which is a good result from binary point of view.

Yet, a naive implementation of this algorithm may still encountered issues related to the inversion of the *constraint Hessian* (which will be defined just after). Currently the paper [15] claims that

- A rounding strategy on x with a common denominator of $2^{O(L)}$ is sufficient.
- This leads to a Hessian $\nabla_x^2 G$ which could be problematic but which can also be rounded with a common denominator of $2^{O(L)}$.
- Finally, computing a $2^{-O(L)}$ approximation of $(\nabla_x^2 G)^{-1}(\nabla_x G)$ can be done in $\widetilde{O}(M^\omega L)$.

Yet the two last claim should be debated. Let introduce the *constraint Hessian* H as the hessian related to $-\sum_m \log(A_m x - b_m)$ (which is common both for log barrier and path following). So,

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

One could remark that $H = A^T \text{Diag}(Ax - b)^{-2} A$.

[15] claims that each $A_m x - b_m$ is in $[2^{-O(L)}, 2^O(L)]$ and can probably be rounded with a $2^{-O(L)}$ common denominator (like to x). But, it is not clear that $\frac{1}{A_1 x - b_1}, ..., \frac{1}{A_M x - b_M}$ will also have a small common denominator. Indeed, in worse case, it seems that the denominator should be as large as the product of numerator of $A_1 x - b_1, ..., A_M x - b_M$ leading to a bound of $2^{O(ML)}$ (product of M numbers bounded by $2^{O(L)}$).

Currently, $\forall i \; H_{i,i} = \sum_m \frac{A_{m,i} A_{m,i}}{(A_m x - b_m)^2} \geq 2^{-L}$. So, one can claim that it is possible to round the matrix as the diagonal is not too small. However, even if, the diagonal contains only term higher than 2^{-L}, approximating a value $H_{i,j}$ with only L digits may lead to put a 0 somewhere there would have been 2^{-ML}. And, this may be an issue if 2 rows/columns are very close i.e. if the matrix is poorly conditioned. Yet, it can be. Currently, H can even be singular if A has not be pre processed such that $Ker(A) = \{x, Ax = \mathbf{0}\} = \{\mathbf{0}\}$ because if 2 variables are exactly the same (i.e. 2 columns of A are equal), then there is a singularity which does not raise issue in solving $Hw = -(\nabla_x G)$ but which still stresses that the matrix H may be poorly conditioned. Thus, claiming that this matrix can be rounded is not trivial.

Precisely, this paper does not claim that statements from [15] are false, but, only that they are not trivial, and, thus hard to implement. Inversely, in a naively implementation, H will have value requiring ML digits (in worse case), and, computing the exact inverse will thus require $M^\omega M^2 L$ binary operations. Indeed, even if the resulting value will finally be rounded on L digits, rounding before the Newton step is not trivially equivalent as rounding after the step.

1.3.4 Summary

Finally, Chubanov algorithm [4] 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 step is ML. The situation seems the same for the *Rescaling Phase* of [13].

So, the binary complexity of most algorithm of the state of the art is significantly higher than $L\times$ the arithmetic complexity: there is an irreducible extra factor M^2 for ellipsoid method [8], M for [4, 13] and $\sqrt{M}$ factor for log barrier [12]. And, at the top of these irreducible extra factors, there is also the issue of what is the exact complexity of solving a linear system with precision at least $2^{-O(L)}$ when each values are bounded by 2^B. There is also finally the question of the possibility to round values $H_{i,j} = \sum_m \frac{A_{m,i}A_{m,j}}{(A_m x - b_m)^2}$ on L digits $(\forall i, j)$ without damaging the resolution of the linear system H (while the exact common denominator could require ML digits in worse case).

As pointed in 1.3.3, [15] claims that $\sum_m \frac{A_{m,i}A_{m,j}}{(A_m x - b_m)^2}$ can be round on L digits, and, that $M^\omega L$ binary operation are sufficient to solve approximately the related system. Yet, this is not trivial. So, instead, this paper considers naive implementation setting in which rounding is only considered at the end of each step. In this setting, $\sum_m \frac{A_{m,i}A_{m,j}}{(A_m x - b_m)^2}$ should be encoded in ML digits, and, computing the inverse of a matrix whose values are bounded by 2^B requires $M^\omega MB$ binary operations and not $M^\omega B$ (as pointed in 1.2).

Thus, under this naive implementation setting, the complexity is increased for all methods based on matrix inversion by M with an additional M when the matrix is a constraint Hessian H [12, 15]. Yet, one can feel that this factor highlights the practical hardness linked with matrix inversion, and, related the numerical instability. Anyway, this naive implementation setting offers a very different view of the state of the art summarized by tables 1 and 2 where the gap between log barrier and ellipsoid algorithm is reduced.

Algorithm	complexity	binary issues
[6]	exponential	none
[8]	$\widetilde{O}(M^4L)$	require ML precision
[13]	$\widetilde{O}(M^3\sqrt{M}L)$	ML halving
[4]	$\widetilde{O}(M^\omega ML)$	ML halving + matrix inversion
[2]	$\widetilde{O}(M^\omega\sqrt{M}L)$	$\sqrt{M}L$ halving + log barrier matrix inversion
[15]	$\widetilde{O}(M^\omega\sqrt{M}L)$	log barrier matrix inversion

Simplex algorithms [6] is exponential but it binary complexity is $L\times$ arithmetic complexity because each step is just the inversion of A which has total binary size L. Inversely, for other algorithms, there are extra factors between binary complexity and arithmetic complexity. Those factors can be irreducible like when a variable is halved ML times (leading to a binary size of ML), or, depend on some precise linear algebra questions.

Table 1: Binary issues in different algorithms from the state of the art.

Advancing on this observation, this paper introduces a new algorithm related to interior point algorithms but with some differences which make the algorithm more stable from binary point of view. Currently, it requires $\widetilde{O}(ML)$ steps (against $\widetilde{O}(\sqrt{M}L)$ for [15]). But, the matrix to inverse has all values naively bounded by 2^L (against 2^{ML} for [15, 12]) and all variables can be trivially rounded on integer. Independently, this algorithm seems like a self concordant version of the Perceptron [16] but disconnected from [13].

Algorithm	arithmetic complexity	naive binary complexity
[16]	exponential	exponential
[6]	exponential	exponential
[8]	$\widetilde{O}(M^4L)$	$\widetilde{O}(M^6L^2)$
[13]	$\widetilde{O}(M^3\sqrt{M}L)$	$\widetilde{O}(M^4\sqrt{M}L^2)$
[4]	$\widetilde{O}(M^\omega ML)$	$\widetilde{O}(M^\omega M^3L^2)$
[12]	$\widetilde{O}(M^\omega\sqrt{M}L)$	$\widetilde{O}(M^\omega M^2\sqrt{M}L^2)$
[15]	$\widetilde{\mathbf{O}}(\mathbf{M}^\omega\sqrt{\mathbf{M}}\mathbf{L})$	$\widetilde{O}(M^\omega M^2\sqrt{M}L^2)$
This	$\widetilde{O}(M^\omega ML)$	$\widetilde{\mathbf{O}}(\mathbf{M}^\omega\mathbf{M^2L^2})$

The naive binary complexity is stated by considering that summing M fractions with numerator/denominator encoded on L digits produces a ML digits fraction, and, that solving a linear system with a matrix whose values are bounded by 2^B requires $M^\omega MB$ binary operations.

Table 2: Naive binary complexity for state of the art algorithms.

2 The self concordant algorithm

The self concordant algorithm is presented in table 3. Basically, 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 $\Upsilon_A = \sqrt{\max_m A_mA_m^T}$. Yet, an additional 1D optimization $v \leftarrow \sqrt{\frac{M}{v^TAA^Tv}}v$ is required in each step allowing to ensure $v^TAA^Tv = M$. This last property will allow to prove that ceiling v with a common denominator of $\Gamma_A = 1000M\sqrt{M}\Upsilon_A$ allows the convergence of the algorithm (and will also offer a convenient bound on v).

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

- $v^TAA^Tv = M$ is not possible on $\mathbb{Q}$ but $v^TAA^Tv \leq 4M$ is - and it is sufficient
- computing $\lambda_{F_A}^2 = (\nabla_vF_A)^T(\nabla_v^2F_A)^{-1}(\nabla_vF_A)$ is possible, and, thus a 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 $v \leftarrow v - \frac{1}{2}(\nabla_v^2F_A)^{-1}(\nabla_vF_A)$ 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 [11].
If $\Psi(x)$ is a self concordant function (mainly 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 that $\Psi(x_\epsilon) - \Psi^* \leq \varepsilon$ in $\widetilde{O}(\Psi(x_{start}) - \Psi^* + \log\log(\frac{1}{\varepsilon}))$ damped Newton steps. Each step is:

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

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

Precisely, 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 that $\Psi(x_\epsilon) - \Psi^* \leq \varepsilon$. This is the so called second phase with quadratic convergence (i.e. $\log\log(\varepsilon)$ steps lead to a precision ϵ). Importantly, $\lambda_\Psi(x_{phase}) \leq \frac{1}{4} \Rightarrow \Psi(x_{phase}) - \Psi^* \leq \frac{1}{4}$

Hadamard bound and Cramer rule: If there exists x such that $Ax \geq \mathbf{1}$, then, there exists a subset S of row indices and a subset R of variable indices such that $A_S x = \mathbf{1}$, $x_R = \mathbf{0}$ is a not singular linear system whose solution χ verifies $A\chi \geq \mathbf{1}$.

In particular, Cramer rules apply to χ. And, Hadamar bound applies to numerators and denominators from Cramer rules.

So, if there exists x such that $Ax \geq \mathbf{1}$, then, there exists one such x with $\log(x^T x) = \widetilde{O}(L)$ where L is the total binary size of A.

Self_concordant_algorithm(A)

F being symbolically $\frac{1}{2}v^T AA^T v - \sum_m \log(v_m)$

$\Upsilon \leftarrow \sqrt{\max_m A_m A_m^T}$

$v \leftarrow \frac{1}{\Upsilon}\mathbf{1}$; $\Gamma \leftarrow 1000M\sqrt{M}\Upsilon$

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**

    $v \leftarrow \sqrt{\frac{M}{v^T AA^T v}}v$

    $v \leftarrow \frac{1}{\Gamma} \times \text{int}(\Gamma \times v + 1)$

  end if

end while

return v

Table 3: Self concordant Perceptron algorithm.

2.2 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.

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

$F_A(w_k) = \frac{1}{2}(v+t\mathbf{1}_k)^T AA^T(v+t\mathbf{1}_k) - \sum_m \log(v_m) + \log(v_k) - \log(v_k+t) = F_A(v) + tA_kA^Tv + \frac{1}{2}t^2A_kA_k^T - \log(v_k+t) + \log(v_k)$. But, $A_kA^Tv \leq 0$, so $F_A(w_k) \leq F_A(v) + \frac{1}{2}t^2A_kA_k^T - \log(v_k+t) + \log(v_k)$, 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 $\Phi(t) = F_A(v) + \frac{1}{2}t^2A_kA_k^T - \log(v_k+t) + \log(v_k)$. Then, $\Phi'(t) = A_kA_k^Tt - \frac{1}{t+v_k}$ and $\Phi''(t) = A_kA_k^T + \frac{1}{(t+v_k)^2}$ 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.

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

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

□

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

Proof. Considering the function $t \to F_A(t \times v) = \frac{1}{2}v^TAA^Tv \times t^2 - \sum_m \log(v_m) - M\log(t)$ trivially proves that $F_A(v)$ decreases when v is normalized such that v^TAA^Tv goes closer to M. In particular, if $v^TAA^Tv \geq$

$4M$, 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:
Assume $Ax \geq \mathbf{1}$ and $v^T AA^T v \leq 4M$, then

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

Proof. Lemma 5 proves that $F_A(v) - F_A^* \leq \frac{1}{16} \Rightarrow v^T AA^T v \leq 4M$. Combined with lemma 1, it leads to $v \leq 2\sqrt{2}||x||_2\mathbf{1}$.

Thus, this bound can be injected in lemma 4 (plus $A_m A_m^T \leq \Gamma^2$ by definition) leading to the conclusion. □

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 $F_A(v) - F_A^* \leq \frac{1}{4Mx^T x\Upsilon^2+1}$. 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 with applies to F which has a minimum (lemma 3 as $X_A \neq \emptyset$) with $\varepsilon = \frac{1}{4Mx^T x\Upsilon^2+1}$.
Yet, this value leads to a solution of the original linear feasibility problem from lemma 6.
For the second point, it is required to use the Cramer rules+Hadamard bound theorem which allows to bound $\log(x^T x)$ by $\widetilde{O}(L)$. 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, $-F^* \leq M\log(x^T x) = O(ML)$ 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 $O(\log\log(4Mx^T x\Upsilon^2 + 1))$ which is just $\log(L)$ steps (definition of L + bound on x + lemma 2). □

Theorem 2:
Assume that $v^T AA^T v \leq 4M$, 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 $F(v + \varpi) \leq F(v) + \frac{1}{2}\varpi^T AA^T \varpi + \varpi^T AA^T v$.

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 $||A^T \varpi||^2 = \varpi^T AA^T \varpi \leq \frac{1}{(1000)^2 M}$.

So, $\varpi^T AA^T v \leq \sqrt{\varpi^T AA^T \varpi \times v^T AA^T v} \leq \sqrt{\frac{1}{(500)^2 M} \times 4M} \leq \frac{1}{250}$ (from Cauchy). And, $\frac{1}{2}\varpi^T AA^T \varpi = \leq \frac{1}{2\times(1000)^2 M} \leq \frac{50}{1000}$. 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 offered rounding scheme correspond to add $\varpi \in \left[0, \frac{1}{\Gamma_A}\right]^M$. □

Theorem 3:
The self concordant described in table 3 always terminates in less then $\widetilde{O}(ML)$ steps eventually returning v such that $AA^T v > \mathbf{0}$. During all the algorithm, all values of v have a common denominator of Γ, and, all numerators are bounded by $2^{O(L)}$.
Finally, $\nabla_v^2 F = AA^T + \text{Diag}(v)^{-2}$. 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 leading to $\mathcal{H}_{i,j} \leq 2^L$.

Proof. First, the second part of the theorem is directly implied by the first 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).

Then, first part of theorem 3 is basically theorem 1+2. But it should be proven carefully. One, it is important to stress that second phase only last $\log(L)$ steps. So, even if a variable is scaled twice during $\log(L)$, then, in fine the variable is only multiplied by L. So binary properties (binary size, bound) will be remain unaffected during second phase because this phase only last $\log(L)$.

This is why the offered algorithm does the ceiling only during first phase i.e. when $\lambda > \frac{1}{4}$. Let stress $v^T AA^T v \geq 4M$ would never happen during second phase, because, at this point $F(v) - F^* \leq \frac{1}{4}$. Thus, even if the scaling of v was not in the *if*, this will be useless during second phase.

Now, during the first phase, each Newton step decreases F by at least $\frac{1}{50}$ ($\frac{1}{100}$ as only 2 approximation of the damped factor will be possible). Yet, the scaling will not increase F, but, will ensure $v^T AA^T v \leq 4M$, and thus, the theorem 2 holds. So, the ceiling phase will not increase F by more than $\frac{1}{200}$. So, the decrease provided by the Newton step is at least twice the increase of the ceiling.

Thus, only 4 modified Newton (2 for the 2 approximation of $\frac{1}{1+\lambda}$, 2 for compensating the ceiling error) steps are required to get (at least) the same effect of 1 raw Newton step. Yet, thank to the ceiling, there is a common denominator Γ for all v values. And, due to lemma 6, numerator are thus bounded by $2\sqrt{Mx^T x}\Gamma$ which is $\widetilde{O}(2^L)$.

Then, as 4 modified Newton offers (at least) the same effect than 1 raw Newton step, theorem 1 proves that termination requires no more than $\widetilde{O}(ML)$ steps. □

2.3 Conclusion

So, the self concordant Perceptron converges in $O(ML)$ steps, with a common denominator and all numerators of it internal state v requiring only L digits each. And, each step is (mainly) the inversion of $\mathcal{H} = \Gamma^2(\text{Diag}(v)AA^T\text{Diag}(v)+I)$ which is an integer matrix with each coefficient bounded by $2^{O(L)}$, while, the constraint matrix $H = A^T\text{Diag}(Ax-b)^{-2}A$ naively requires ML digits per value. Indeed, H should naively be

scaled as $\prod_m \text{numerator}(A_m x - b_m)^2 \times A^T \text{Diag}(Ax-b)^{-2} A$ to recover an integer matrix.

To conclude, the contribution of this paper is a new algorithm whose naive binary implementation will have better performance than all naive binary implementations of all state of the art algorithms (in particular naive self concordant Perceptron outperforms naive path following [15] by a factor $\sqrt{M}$ thank to a much lighter matrix to inverse). Inversely, expert implementation of path following (whose theoretical existence is claimed by [15]) may outperforms expert implementation of self concordant Perceptron by a factor $\sqrt{M}$. Yet, the offered algorithm can thus be relevant for critical applications where both binary issue is unacceptable, and, expert implementation hard to prove using common tool for formal verification.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] George B et. al. Dantzig. The generalized simplex method for minimizing a linear form under linear inequality restraints. In *Pacific Journal of MathematicsAmerican Journal of Operations Research*, 1955.

[7] 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.

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

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

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

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

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

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

[14] 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.

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

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

This paper is an updated version of [3] which has an hal-preprint at hal-02491694. This previous version contains an appendix providing more detail on pre and post processing required for linear feasibility. But, the proof is improved in this version. And, a more complete description of the binary issues related to state of the art is presented in this version.