

Solving Linear Programming While Tackling Number Representation Issues

Adrien Chan-Hon-Tong ^a
ONERA, Université Paris Saclay, France

Keywords: Linear Programming, Polynomial Time Complexity, Number Representation.

Abstract: Gaussian elimination is known to be exponential when done naively. Indeed, theoretically, it is required to take care of the intermediary numbers encountered during an algorithm, in particular of their binary sizes. However, this point is weakly tackled for linear programming in state of the art. Thus, this paper introduces a new polynomial times algorithm for linear programming focusing on this point: this algorithm offers an explicit strategy to deal with all number representation issues. One key feature which makes this Newton based algorithm more compliant with binary considerations is that the optimization is performed in the so-called first phase of Newton descent and not in the so-called second phase like in the state of the art.

1 INTRODUCTION

Linear programming is a central optimization problem which aims to solve:

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

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 .

Today, the state of the art is central-path log-barrier (Nesterov and Nemirovskii, 1994) and/or path-following (Renegar, 1988) algorithms which solves linear programs with total binary size L in less than $\tilde{O}(\sqrt{ML})$ Newton steps (assuming $N = O(M)$). As each Newton step is mainly the resolution of an $M \times M$ linear system, the arithmetic time complexity of those algorithms is $\tilde{O}(M^\omega \sqrt{ML})$ ($\tilde{O}(\cdot)$ notation will be used instead of $O(\cdot)$ to express the fact that log factors are omitted) where ω is the coefficient of matrix multiplication i.e. 3 with simple algorithm but 2.38 with (Ambainis et al., 2015). Faster randomized algorithms like (Cohen et al., 2021) are not in the scope of this paper.


However, considering that matrix inversion can be done in $\tilde{O}(M^\omega)$ operations is only half of the story. It is true by considering operation on \mathbb{Z} or \mathbb{Q} as 1 operation. But, from theoretical point of view, either operations are realized with fixed precision (opening the

door to numerical instabilities), or, all numbers have to be represented either by an arbitrary large integer or a fraction of arbitrary large integer. But, in this setting, it is required to count the number of **binary** operations. Even more, it is required to take care of the binary sizes of the number that appear during the algorithm to avoid bad binary time complexity like in naive Gaussian elimination which is exponential for this reason (Fang and Havas, 1997).

Precisely, a binary-compliant algorithm for linear programming should have all the 3 following features:

- a good arithmetic time complexity
- an explicit rounding strategy for the intermediary numbers
- which maintains the binary size of those numbers bounded by $\tilde{O}(L)$

Typically, Simplex-like algorithms (Dantzig, 1955) have no binary issues because the intermediary points are related to a linear subsystem of the input constraints but they have not a good arithmetic time complexity. Inversely, most state of art methods rely on scaling, i.e. at some point in the algorithm, there is an operation like $A = A(I + xx^T)$ in (Peña and Soheili, 2016), or $column(A, k) = \frac{1}{2} \times column(A, k)$ in (Chubanov, 2015) or $\mu = \frac{1}{2} \times \mu$ in (Anderson et al., 1996) (a classical implementation for central path log barrier). Yet, structurally, even if one could set up an explicit rounding strategy, those algorithms will not respect the property of having intermediary numbers

^a  <https://orcid.org/0000-0002-7333-2765>

bounded by $\tilde{O}(L)$ as one variable scaled twice each of the \sqrt{ML} steps is finally scaled by $2^{\sqrt{ML}}$.

Currently, from state of the art algorithms, only (Renegar, 1988) has both the first and the third features because in (Renegar, 1988) scaling is performed with a factor $(1 + \frac{O(1)}{\sqrt{M}})$ ($(1 + \frac{O(1)}{\sqrt{M}})^{\sqrt{ML}} \approx 2^{\log(e)L}$) so this scaling still leads to a binary size of $\tilde{O}(L)$. Yet, (Renegar, 1988) only offers a sketch of strategy to round intermediary numbers, and, admits that it sketch depends on several implicit constants. Thus, there is not known algorithm with the 3 features allowing to be fully binary-compliant.

Yet, this paper introduces a new algorithm related to interior point algorithms but with those 3 features. Currently, it requires $\tilde{O}(ML)$ steps against $\tilde{O}(\sqrt{ML})$ for (Renegar, 1988; Nesterov and Nemirovskii, 1994). But, it is scaling free with variable sizes naturally bounded by $\tilde{O}(L)$. And, the internal variables can be explicitly rounded to integer at the end of each Newton steps (potentially a final phase is required but this final phase only last at most $\tilde{O}(\log(L))$ so it is negligible).

This situation is summarized by Table 1.

Finally, this algorithm also takes care of rounding/approximating operations which could lead to irrational numbers (typically $\sqrt{2}$ has no binary number representation, so rounding should be used instead), as, the main interest of this paper is to tackle number representation issues. Independently, the offered algorithm can be seen as a self concordant version of Perceptron (Rosenblatt, 1958) and/or (Peña and Soheili, 2016), it is faster than (Peña and Soheili, 2016) which requires $\tilde{O}(M^2\sqrt{ML})$ Perceptron steps i.e. $\tilde{O}(M^4\sqrt{ML})$ arithmetic operations.

2 SELF CONCORDANT PERCEPTRON

The offered algorithm is described in table 2.

Currently, this algorithm solves **linear feasibility** a simpler form of linear programming which consists to solve:

$$\text{find } x \in X_A = \{\chi \in \mathbb{Q}^N, A\chi > \mathbf{0}\} \quad (2)$$

for a given matrix $A \in \mathbb{Z}^{M \times N}$ such that $X_A \neq \emptyset$ and $\mathbf{0}$ is the vector with all 0. Trivially, this problem is equivalent to find x such that $Ax \geq \mathbf{1}$ with $\mathbf{1}$ the vector filled by 1. Importantly, not all matrix A could be encountered when solving linear feasibility because the problem assumes $X_A \neq \emptyset$. Typically, the algorithm offered in table 2 may have an undefined behavior if the input matrix A does not verify $X_A \neq \emptyset$.

But, despite this linear feasibility problem may seem very specific, it is theoretically equivalent with general linear programming: some well known strongly polynomial times pre/post processing (mainly based on duality theory) allows to solve any linear programming instance by solving a linear feasibility instance with equivalent size and total binary size and $X_A \neq \emptyset$. For completeness, these well known equivalences are recalled in appendix.

Finally, a simplified version of the algorithm which will be useful for proving the convergence is provided in table 3 (yet it does **not** verify the binary-compliant features). This simplified version allows to see that the offered self concordant Perceptron is just Newton descent on the function

$$F_A(v) = \frac{1}{2}v^T A A^T v - \sum_{m \in \{1, \dots, M\}} \log(v_m) \quad (3)$$

3 PROOFS

3.1 Convergence

Theorem from (Nemirovski, 2004):

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 $\Psi(x_\epsilon) - \Psi^* \leq \epsilon$ in $\tilde{O}(\Psi(x_{start}) - \Psi^* + \log \log(\frac{1}{\epsilon}))$ damped Newton steps. Each step simply relies on two linear algebra operations: $\lambda_\Psi(x) \leftarrow \sqrt{(\nabla_x \Psi)^T (\nabla_x^2 \Psi)^{-1} (\nabla_x \Psi)}$ and $x \leftarrow x - \frac{1}{1 + \lambda_\Psi(x)} (\nabla_x^2 \Psi)^{-1} (\nabla_x \Psi)$. Precisely,

- 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, $\tilde{O}(\log \log(\frac{1}{\epsilon}))$ additional steps are required to get x_ϵ such that $\Psi(x_\epsilon) - \Psi^* \leq \epsilon$. This is the so called second phase with quadratic convergence (i.e. $\log \log(\epsilon)$ steps lead to a precision ϵ).

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, $\frac{\|v\|_2^2}{\|x\|_2^2} \leq \|A^T v\|_2^2$.

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, $\mathbf{1}^T v \leq \|x\|_2 \times \|A^T v\|_2$.

Table 1: Self concordant Perceptron offers both a mastered internal binary size, and, an explicit rounding strategy while having an arithmetic time complexity only higher than the state of the art by a factor \sqrt{M} .

Algorithm	\mathbb{Q} time complexity	internal binary size bounded by $\tilde{O}(L)$	explicit rounding
(Rosenblatt, 1958)	exponential	no	no
(Dantzig, 1955)	exponential	yes	yes
(Peña and Soheili, 2016)	$\tilde{O}(M^4\sqrt{ML})$	no	no
(Anderson et al., 1996)	$\tilde{O}(M^{\omega}\sqrt{ML})$	no	no
(Renegar, 1988)	$\tilde{O}(M^{\omega}\sqrt{ML})$	yes	no
Self concordant Perceptron (this)	$\tilde{O}(M^{\omega}ML)$	yes	yes

 Table 2: Self concordant Perceptron solves linear feasibility in $\tilde{O}(ML)$ steps with mainly integer computations.

Algorithm(A)
1. $\Gamma \leftarrow \text{int}(1000M\sqrt{M\max_m A_m A_m^T}) + 1$
2. $w \leftarrow \left(\text{int}\left(\Gamma\sqrt{\frac{M}{\Gamma^T A A^T \Gamma}}\right) + 1\right) \times \mathbf{1}$
3. if $AA^T w > \mathbf{0}$ return w
4. $H \leftarrow \begin{pmatrix} w_1^2 & 0 & \dots \\ 0 & \dots & 0 \\ \dots & 0 & w_M^2 \end{pmatrix} AA^T + \Gamma^2 \mathbf{I}$
5. $h \leftarrow \Gamma^2 \times w$ $\quad - \begin{pmatrix} w_1^2 & 0 & \dots \\ 0 & \dots & 0 \\ \dots & 0 & w_M^2 \end{pmatrix} AA^T w$
6. solve the integer linear system $H\mathcal{N} = h$ <i>//</i> \mathcal{N} may be fractional
7. $\lambda_2 \leftarrow \mathcal{N}^T h $
8. compute θ , $\frac{1}{2}\theta \leq 1 + \sqrt{\frac{\lambda_2}{(1000)^3 M^4 \sqrt{M} \Gamma^3}} \leq \theta$
9. if $\lambda_2 > \frac{\Gamma^3}{16}$ go to 10 else go to 13
10. $q \leftarrow \text{int}\left(\sqrt{\frac{(w+\theta\mathcal{N})^T A A^T (w+\theta\mathcal{N})}{M\Gamma^2}}\right) + 1$
11. $w \leftarrow \text{int}\left(\frac{v+\theta\mathcal{N}}{q}\right) + \mathbf{1}$
12. go to 3
13. $w \leftarrow v + \theta\mathcal{N}$ <i>// no rounding for the so-called second phase</i>
14. go to 3

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 $v \geq \mathbf{0}$, $v^T v \leq (\mathbf{1}^T v)^2$. \square

Lemma 2:

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

Table 3: A simplified version of the self concordant Perceptron algorithm.

Simplified Algorithm(A)
$\Upsilon = \sqrt{\max_m A_m A_m^T}$
$v \leftarrow \frac{1}{\Upsilon} \mathbf{1}$
while $\neg(AA^T v > \mathbf{0})$ do
$F \leftarrow \frac{1}{2} v^T AA^T v - \sum_m \log(v_m)$
$\mathcal{N} \leftarrow (\nabla_v^2 F)^{-1} (\nabla_v F)$
$\lambda \leftarrow \sqrt{(\nabla_v F)^T \mathcal{N}}$
$v \leftarrow v - \frac{1}{1+\lambda} \mathcal{N}$
end while
return v

Proof. f is a continuous function from $]0, \infty[$ to \mathbb{R} . $f(t) \rightarrow \infty$ due to the $-\log$, and, $f(t) \rightarrow \infty$ 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 $f^* = f(\|x\|_2)$. \square

Lemma 3:

Assume $X_A \neq \emptyset$, F_A is lower bounded - with F_A the function defined in equation (3) i.e.

$$F_A(v) = \frac{v^T A A^T v}{2} - \sum_m \log(v_m)$$

Proof. If $X_A \neq \emptyset$, 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 $f(t) \rightarrow \infty$ 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 $F_A^* \geq Mf^*$. \square

From now, the assumption that $X_A \neq \emptyset$ will be omitted.

Remark: As F_A (defined in eq.(3)) has a minimum when $X_A \neq \emptyset$ (lemma 3), then, theorem from (Ne-

mirovski, 2004) holds in this case: the damped Newton descent on F_A from any v_{start} will provide v such that $F_A(v) - F_A^* \leq \varepsilon$ in less than $F_A(v_{start}) - F_A^* + \log \log(\frac{1}{\varepsilon})$.

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

$F_A(w_k) = \frac{1}{2}(v + t \mathbf{1}_k)^T AA^T (v + t \mathbf{1}_k) - \sum_m \log(v_m) + \log(v_k) - \log(v_k + t) = F_A(v) + t A_k A_k^T v + \frac{1}{2} t^2 A_k A_k^T - \log(v_k + t) + \log(v_k)$. But, $A_k A_k^T v \leq 0$, 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. \square

Lemma 5:

Assume $Ax \geq \mathbf{1}$, then $F_A(v) \leq F_A(v_{start}) \Rightarrow v \leq \|x\|_2 \times (1 + \|x\|_2 (F_A(v_{start}) + M \log(\|x\|_2))) \times \mathbf{1}$.

Proof. From lemma 2, $F_A(v) \geq \sum_m f(v_m)$ with $f(t) = \frac{t^2}{2\|x\|_2^2} - \log(t)$. Thus, for all k , $F_A(v) \geq f(v_k) + (M - 1)f^*$ (and $f^* = -\log(\|x\|_2^2)$ see lemma 2). So $F(v) \leq F_A(v_{start}) \Rightarrow f(v_k) \leq F(v_{start}) + M \log(\|x\|_2^2)$.

But, $\forall t \geq t^*, \exists \theta, f(t) = f(t^*) + (t - t^*)f'(t^*) + \frac{1}{2}(t - t^*)^2 f''(t^*) + \frac{1}{6}(t - t^*)^3 f'''(\theta)$. Yet $f'''(t) < 0$ and $f'(t^*) = 0$, so it holds that $f(t) \geq \frac{1}{2}(t - t^*)^2 f''(t^*) = \frac{(t - t^*)^2}{\|x\|_2^2}$.

So $f(t) \leq F_A(v_{start}) + M \log(\|x\|_2) \Rightarrow t \leq t^* + \|x\|_2 \sqrt{F_A(v_{start}) + M \log(\|x\|_2)}$. Applying this last inequality to each component of v provide the expected inequality. \square

Theorem 1:

Damped Newton descent on F_A starting from any v_{start} will terminate eventually returning v such that $AA^T v > \mathbf{0}$ - precisely this

will happen at least when $F_A(v) - F_A^* \leq \min_m \frac{1}{(\|x\|_2^2 \times (1 + \|x\|_2 (F(v_{start}) + M \log(\|x\|_2)))^2 A_m A_m^T + 1)}$

Proof. This is just lemma 4 + lemma 5 \square

3.2 Complexity

Definition for the rest of this paper,

$$\Upsilon_A = \sqrt{\max_m A_m A_m^T} \quad (4)$$

In particular, $F_A(\frac{1}{\Upsilon_A} \mathbf{1}) \leq M^2 + M \log(\Upsilon_A)$ (from Cauchy inequality).

Theorem from (Khachiyan, 1979):

There are standard complexity results about the link between the total binary size L and the determinant of a matrix since (Khachiyan, 1979). Typically, assuming $X_A \neq \emptyset$, there exists x such that $Ax \geq \mathbf{1}$. Even more, one such x can be expressed as linear system extracted from A . So, $\log(\|x\|_2^2) = \tilde{O}(L)$ where L is the total binary size of input matrix A . Also, trivially, $\log(\Upsilon_A) = \tilde{O}(L)$.

Theorem 2:

The simplified self concordant Perceptron presented in table 3 solves linear feasibility in less than $\tilde{O}(ML)$ Newton steps.

Importantly, the so-called second phase of the Newton descent is negligible.

Proof. The proof is mainly the theorem from (Nemirovski, 2004) with the correct value for v_{start} , F_A^*, ε .

First, theorem 1 provides the correct value for ε (such that the algorithm outputs v with $AA^T v > \mathbf{0}$). Now, the ε is into a double log resulting in negligible $\tilde{O}(\log(L))$ complexity.

Thus, contrary to the state of the art, almost all the algorithm takes place in the so called first phase which lasts $\tilde{O}(F_A(v_{start}) - F_A^*) = \tilde{O}(ML)$ damped Newton steps. Now, $v_{start} = \frac{1}{\Upsilon_A} \mathbf{1}$, so $F_A(v_{start}) = M^2 + M \log(\Upsilon_A) = \tilde{O}(ML)$ and $-F^* \leq M \log(\|x\|_2^2) = \tilde{O}(ML)$ from definition of Υ_A , and, lemma 2, and, from standard complexity results. \square

At this point, this paper proves that the simplified self concordant Perceptron (table 3) solves linear feasibility in $\tilde{O}(ML)$ damped Newton steps. However, this result is not really interesting by itself: it is almost a corollary of self concordant theory from (Nemirovski, 2004), and, better algorithms exists (requiring only $\tilde{O}(\sqrt{ML})$ steps e.g. (Nesterov and Nemirovskii, 1994)). Yet the interesting point of the paper is the binary property of the complete self concordant Perceptron (table 2), and, proven bellow.

3.3 Binary Property

Lemma 6:

In damped Newton step, $\frac{1}{1+\lambda}$ can be replace by a 2 approximation.

Proof. F_A is convex so $F_A(v - \theta(\nabla_v^2 F_A)^{-1}(\nabla_v F_A)) \leq \frac{1}{2}(F(v) + F_A(v - \frac{1}{1+\lambda(v)}(\nabla_v^2 F_A)^{-1}(\nabla_v F_A)))$ if $\frac{1}{2} \frac{1}{1+\lambda(v)} \leq \theta \leq \frac{1}{1+\lambda(v)}$ \square

Lemma 7:

$F_A(\sqrt{\frac{M}{v^T A A^T v}} \times v) < F_A(v)$ - or with integer -
 $F_A\left(\frac{1}{\text{int}(\sqrt{\frac{v^T A A^T v}{M}})+1} v\right) \leq F_A(v)$

Proof. Considering the function $t \rightarrow F_A(t \times v)$ trivially proves that $F_A(v)$ decreases when v is normalized such that $v^T A A^T v$ goes closer to M .

The rounded version $\frac{1}{\text{int}(\sqrt{\frac{v^T A A^T v}{M}})+1}$ only manipulate integer and guarantees that $v^T A A^T v \in [\frac{M}{4}, 4M]$. \square

Definition for the rest of this paper,

$$\Gamma_A = 1000M\sqrt{M}\Upsilon_A \quad (5)$$

Theorem 3:

Assume that $v^T A A^T v \leq 4M$, then:

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

In particular, $\forall v$,

$$F\left(\frac{\text{int}(\Gamma_A \times v_1)+1}{\Gamma_A} \dots \frac{\text{int}(\Gamma_A \times v_M)+1}{\Gamma_A}\right) \leq F(v) + \frac{1}{200}$$

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

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

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

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

Theorem 4:

The self concordant Perceptron presented in table 2 is consistent with the simplified version (table 3), and, adds:

- normalizing of v such that $v^T A A^T v \leq 4M$
- 2 approximation of $\frac{1}{1+\lambda}$
- rounding all components of v with a common denominator of Γ_A
- and with w being the numerator after rounding (could be seen as substitution $w = \Gamma_A \times v$)

and, thus, this algorithm converges like the simplified version presented in table 3.

Proof. First, normalizing v is not an issue as it decreases F_A as proven in lemma 7.

Then, as recalled in theorem from (Nemirovski, 2004), each damped Newton step (during the so called first phase) decreases F_A by at least $\frac{1}{50}$. Approximating of $\frac{1}{1+\lambda}$ adds a factor $\frac{1}{2}$ (see lemma 6).

Yet, as pointed in theorem 3, rounding on a common denominator of Γ_A only increases F_A by $\frac{1}{200}$.

So, combining damped Newton step with approximation of $\frac{1}{1+\lambda}$ + normalization + rounding decreases F_A by at least $\frac{1}{200} = \frac{1}{50} \times \frac{1}{2} - \frac{1}{200}$ ($\frac{1}{50}$ for the original damped Newton step, $\frac{1}{2}$ due to the approximation, $-\frac{1}{200}$ due to the rounding). From complexity point of view, the conclusion is that the complete step of self concordant Perceptron decreases F_A by $O(1)$.

Finally, to give an explanation of the steps 4 and 5 of algorithm table 2, $\nabla_v F_A = A A^T v - \begin{pmatrix} \frac{1}{v_1} \\ \dots \\ \frac{1}{v_M} \end{pmatrix}$ So $(\Gamma_A)^3 \begin{pmatrix} v_1 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & v_M \end{pmatrix}^2 (\nabla_v F_A) = -h$ And, $\nabla_v^2 F_A = A A^T + \begin{pmatrix} \frac{1}{v_1} & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & \frac{1}{v_M} \end{pmatrix}^2$, so,

$$(\Gamma_A)^2 \begin{pmatrix} v_1 & \dots & 0 \\ 0 & \dots & 0 \\ 0 & \dots & v_M \end{pmatrix}^2 (\nabla_v^2 F_A) = H$$

This way, \mathcal{N} computed in step 6 of algorithm table 2 is the Newton direction adapted to $w = \Gamma \times v$.

So, the self concordant Perceptron from table 2 has exactly the same property than the simplified version of table 3. \square

4 CONCLUSIONS

This paper introduces an algorithm for linear programming with arithmetic complexity of $\tilde{O}(ML)$

Newton steps (higher from the state of the art by a factor \sqrt{M}). But, this algorithm has good binary property: it keeps the binary size of intermediary numbers bounded by $\tilde{O}(L)$, and, offers an explicit strategy for rounding all intermediary numbers (see table 1).

REFERENCES

- Ambainis, A., Filmus, Y., and Le Gall, F. (2015). 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.
- Anderson, E. D., Gondzio, J., Mészáros, C., and Xu, X. (1996). Implementation of interior-point methods for large scale linear programs. In *Interior Point Methods of Mathematical Programming*, pages 189–252. Springer.
- Chubanov, S. (2015). A polynomial projection algorithm for linear feasibility problems. *Mathematical Programming*, 153(2):687–713.
- Cohen, M. B., Lee, Y. T., and Song, Z. (2021). Solving linear programs in the current matrix multiplication time. *Journal of the ACM (JACM)*, 68(1):1–39.
- Dantzig, G. B. e. a. (1955). The generalized simplex method for minimizing a linear form under linear inequality restraints. In *Pacific Journal of Mathematics American Journal of Operations Research*.
- Fang, X. G. and Havas, G. (1997). On the worst-case complexity of integer gaussian elimination. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, pages 28–31.
- Khachiyan, L. (1979). A polynomial algorithm for linear programming. *Doklady Akademii Nauk SSSR*.
- Nemirovski, A. (2004). Interior point polynomial time methods in convex programming. *Lecture notes*, 42(16):3215–3224.
- Nesterov, Y. and Nemirovskii, A. (1994). *Interior-point polynomial algorithms in convex programming*. Siam.
- Peña, J. and Soheili, N. (2016). A deterministic rescaled perceptron algorithm. *Mathematical Programming*, 155(1-2):497–510.
- Renegar, J. (1988). A polynomial-time algorithm, based on newton’s method, for linear programming. *Mathematical programming*, 40(1):59–93.
- Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

APPENDIX

Equivalence of Linear Programming and Linear Feasibility

This paper provides an algorithm \mathbf{algo}_0 which returns v such that $AA^T v > \mathbf{0}$ on an input A assuming $\exists x, Ax \geq \mathbf{1}$ (undefined behaviour otherwise - v is positive but this does not matter). Trivially, it is thus possible to form \mathbf{algo}_1 which returns x such that $Ax > \mathbf{0}$ on input A assuming such x exists by returning $A^T \mathbf{algo}_0(A)$ (undefined behaviour otherwise).

- Thank to \mathbf{algo}_1 , one could form $\mathbf{algo}_2(A, b)$ which returns x such that $Ax > b$ assuming such x exists (undefined behaviour otherwise). Indeed, let consider any A, b such that $\exists x, Ax > b$, finding such x is equivalent to find a pair x, t such that $Ax - t \times b > \mathbf{0}$ and $t > 0$, because $\frac{x}{t}$ is then a solution of the original problem. Formally, let \mathcal{A}_1 the matrix A plus $\mathbf{1}$ as additional column and $(\mathbf{0} \ 1)$ as additional row. Thus, one can get $(x_1 \ t_1)$ by computing $\mathbf{algo}_1(\mathcal{A}_1)$ and returning $\frac{x_1}{t_1}$ as output of $\mathbf{algo}_2(A, b)$.

Importantly, only constant number of variables/constraints are added, and, binary size is not increased. So complexity of $\mathbf{algo}_2(A, b)$ is the same than $\mathbf{algo}_1(A, b)$.

- Thank to \mathbf{algo}_2 , one could form $\mathbf{algo}_3(A, b)$ which returns x such that $Ax \geq b$ assuming such x exists. Indeed, if $\exists x/Ax \geq b$, then a fortiori $\exists x, t$ such that $Ax + t\mathbf{1} \times > b$, $0 < t < \frac{1}{\Omega(A)}$ (with $\Omega(A)$ the maximal subdeterminant of A). So, one could call \mathbf{algo}_1 on \mathcal{A}_2, b_2 with \mathcal{A}_2 being A plus $\mathbf{1}$ column plus a row with $\mathbf{0}$ and $\Omega(A)$ and b_2 being b plus two 1. Thus, $\mathbf{algo}_2(\mathcal{A}_2, b_2) = x_2, t_2$.

Now, one could consider greedy improvement of $\min_{x,t, Ax+t\mathbf{1} \geq b, t \geq 0} t$ initialized from (x_2, t_2) . Such greedy improvement can be performed by projecting (x, t) on $\{(x, t), Ax + t\mathbf{1} \geq b\}$ while minimizing t . One such greedy step can simply be done by looking for χ, τ such that $A_S \chi + t\mathbf{1}_S = 0$ and $\tau = -1$ with S the saturated rows in $Ax + t\mathbf{1} \geq b$. If no such χ, τ exists, the greedy improvement has terminated otherwise one could do $(x, t) \leftarrow (x + \mu\chi, t + \mu\tau)$ with μ such that $Ax + t\mathbf{1} \geq b, t \geq 0$. There will be no more than M such greedy purification because one row enter the saturated ones at each step.

When this greedy process terminates, this leads to \hat{x}, \hat{t} with $A\hat{x} + \hat{t}\mathbf{1} \geq b$, $0 \leq \hat{t} \leq t_2 < \frac{1}{\Omega(A)}$ but \hat{x}, \hat{t} is a vertex of A . So Cramer rule applies, and so $\hat{t} =$

$\frac{Det(S_t)}{Det(S)}$ with S a sub matrix of A and S_t the Cramer partial submatrix related to t . But $\hat{t} \leq t_2 \leq \frac{1}{\Omega(A)}$, so $\hat{t} = 0$, and thus, $A\hat{x} \geq b$. So, this projection of x_2, t_2 gives $x_3 = \mathbf{algo}_3(A, b)$.

Importantly, the binary size of \mathcal{A}_2, b_2 is just twice the binary size of A, b because $\log(\Omega(A)) \leq L(A)$, so $L(\mathcal{A}_2) = L(A) + \log(\Omega(A)) \leq 2L(A)$ and only a constant number of variables constraints are added. so complexity of $\mathbf{algo}_3(A, b)$ is the same than $\mathbf{algo}_2(A, b)$.

- Let any A, b - **without assumption** - solving $Ax \geq b$ (or producing a certificate that no solution exists) is equivalent to solve $\min_{z / Az+t\mathbf{1} \geq b, t \geq 0} t$ (there is a solution if the minimum is 0). Yet, this last linear program is structurally feasible ($x = 0$ and a sufficiently large t provided a feasible point) and bounded because $t \geq 0$. Thus, primal dual theory gives a system $A_{primal-dual}(x \ y) \geq b_{primal-dual}$ whose solution contains solution of the linear program $\min_{z / Az+t\mathbf{1} \geq b, t \geq 0} t$.

Applying $\mathbf{algo}_3(A_{primal-dual}, b_{primal-dual})$ provides thus such $x_{primal-dual}, y_{primal-dual}$ from which one could restore x_3, t_3 with either $t_3 = 0$ and so $Ax_3 \geq b$ or $t_3 \neq 0$ (this is a certificate).

This leads to an algorithm \mathbf{algo}_4 which is able to find x such that $Ax \geq b$ (or to produce a certificate that no solution exists) without assumption on A, b about the existence or not of such x .

Importantly, the number of variables-constraints is only scaled two folds when computing the primal dual, so from theoretical point of view, it does not change the complexity between \mathbf{algo}_4 and \mathbf{algo}_3 .

- Finally, for any A, b, c without any assumption solving $\min_{x / Ax \geq b} c^T x$ can be done with 2 \mathbf{algo}_4 calls and one \mathbf{algo}_3 call:
 - one to know if the problem is feasible i.e. $\mathbf{algo}_4(A, b)$
 - one on the dual to know if it is bounded $\mathbf{algo}_4(A_{dual}, b_{dual})$
 - and one call to \mathbf{algo}_3 on the primal dual to get the optimal solution (if previous two computations certify that the problem is feasible and bounded).

Again, from theoretical point of view, the complexity does not change: it only does 3 calls on instances only scaled 2 times. At the end, it returns the optimal solution or a certificate that the problem is not feasible or not bounded.

Thus, an algorithm \mathbf{algo}_0 which returns v such that $AA^T v > \mathbf{0}$ on input A if there exists such v (undefined behaviour otherwise) allows to build with same complexity $\mathbf{algo}_5(A, b, c)$ which solves general linear programming.

The opposite way is trivial $\mathbf{algo}_5(AA^T, \mathbf{1}, \mathbf{0})$ is a correct implementation of $\mathbf{algo}_1(A)$ for any A .

Table 4: The sequences of pre/post processing connecting linear feasibility and linear programming.

Assume $\mathbf{algo}_1(A)$ takes A and returns one x such that $Ax > \mathbf{0}$ if one exists, then:

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

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

return $xt[: -1]/xt[-1]$ //python convention

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

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

$\Gamma \leftarrow$ Hadamard bound on A

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

$$x_2, t_2 \leftarrow xt[: -1], xt[-1]$$

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

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.

$\mathbf{algo}_4(A, b)$

$$A_p \leftarrow \begin{pmatrix} A & \mathbf{1} \\ 0 & 1 \end{pmatrix}, b_p \leftarrow \begin{pmatrix} b \\ 0 \end{pmatrix}, c_p \leftarrow (\mathbf{0} \ 1)$$

compute $A_{dual}, b_{dual}, c_{dual}$ with duality theory

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

$$x \leftarrow \chi[: M], t \leftarrow \chi[M]$$

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 $\chi[: M]$

end if

is a standard linear programming solver.