An interesting link between linear feasibility, linear programming, support vector margin and convex hull.

Adrien Chan-Hon-Tong

An interesting link between linear feasibility, linear programming, support vector margin and convex hull.

Adrien CHAN-HON-TONG
ONERA - université paris sud

April 12, 2019

Abstract

This short paper presents an algorithm based on simple projections and linear feasibility ($Ax = \mathbf{0},\ x > \mathbf{0}$) queries which solves both linear programming ($\min_{x\ /\ Ax \geq b} cx$), and, support vector margin ($\min_{w\ /\ Aw \geq 1} w^T w$).

In addition, this algorithm is strongly polynomial on special instance of both linear program and support vector margin, characterized by a similar convex hull property.

Thus, this algorithm could be interesting as a link between all these four notions: linear feasibility, linear programming, support vector margin and convex hull.

1 Introduction

Linear programming is the very studied task of solving $\min_{x\ /\ Ax \geq b} c^T x$ with A a matrix, b and c some vectors. This problem can be solved in polynomial time since [15, 8, 13]. State of the art algorithm to solve linear program are today interior point algorithms (e.g. [18]). In addition, several special case of linear program can be solved more efficiently:

- combinatorial linear program [21]
- linear program with at most two variables per inequality [11]
- markov chain [20]
- system having binary solution [4]

Support vector margin [6] used to be a central tool of machine learning (before deep learning [16]), and, consist to solve $\min_{w\ /\ Aw \geq 1} w^T w$. Complexity of support vector margin seems to be polynomial as a special problem of semi

definite programming complexity [9], although semi definite complexity without assumption under degeneracy is not that clear [10] (when *solving* is about producing an exact solution plus a corresponding certificate). And, in addition, approximate solutions for support vector margin exist like [12] (theoretically proven) or [14, 3] (efficient implementations).

Independently, [5] recently shows that linear feasibility i.e. $Ax = \mathbf{0},\ x > \mathbf{0}$ can be solve in strongly polynomial time. This result is especially interesting, because [2] shows that major interior point family does not solve linear program in strong polynomial time. And, so, the question of solving linear program as a strong polynomial sequence of linear feasibility queries is relevant.

This short paper focus on this last point, and, presents, in section 2, an algorithm which solves both linear programming and support vector margin by solving a sequence of linear feasibility on maximal subset. Although, the complexity of this algorithm may be exponential, it is showed, in section 3, that special instance of both linear programming and support vector margin can be solved in strong polynomial time under assumptions linked to convex hull properties.

Thus, this short paper provides at least an interesting links between linear feasibility, linear programming, support vector margin and geometric complexity of convex hull, which is a very studied subject (e.g. [1, 17, 19]).

Notations

$\mathbb{N}, \mathbb{Q}$ are the sets of integer and rational numbers. $\setminus$ is the ensemble subtraction. $\forall n \in \mathbb{N}\setminus\{0\}$, $R(n)$ is the range of integer from 1 to n i.e. $R(n) = \{k \in \mathbb{N}, 1 \leq k \leq n\} = \{1, ..., n\}$.

$\forall I, J \in \mathbb{N}\setminus\{0\}$, $\mathbb{Q}^I$ is the set of I dimensional vector of $\mathbb{Q}$, and, $\mathbb{Q}_{I,J}$ is the set of matrix with I rows and J columns, with values in $\mathbb{Q}$. $\mathbb{Q}^I$ would be matched with $\mathbb{Q}_{I,1}$ i.e. vector are seen as a column vector.

$\forall i \in \mathbb{N}\setminus\{0\}$, $._i$ designs the i component: a rational for vector or a row for a matrix. Rows are seen as row vector i.e. if $\forall A \in \mathbb{Q}_{I,J}$, $A_i \in \mathbb{Q}_{1,J}$. Also ^{T} is the transposition operation i.e. $\forall I, J \in \mathbb{N}\setminus\{0\}$, $\forall A \in \mathbb{Q}_{I,J}$, $A^T \in \mathbb{Q}_{J,I}$ with $\forall (i,j) \in R(I) \times R(J)$, $A^T_{j,i} = A_{i,j}$. For all set $S \subset \mathbb{N}$, A_S, b_S is the submatrix or subvector obtained when keeping only rows or components $s \in S$.

$\mathbf{0}$ and $\mathbf{1}$ are the 0 and 1 vector i.e. vector contains only 0 or only 1, and $\mathbf{I}$ is the identity matrix.

$\mathbb{U}_I$ is the set of normalized vector from $\mathbb{Q}^I$ i.e. such that $v^T v = 1$ i.e. $\mathbb{U}_I = \{v \in \mathbb{Q}^I, v^T v = 1\}$. $\mathbb{U}_{I,J}$ is the set of matrix from $\mathcal{M}_{I,J}$ whose rows (after transposition) are in $\mathbb{U}_J$ (only the rows, not necessarily the columns).

Also, if $A \in \mathcal{Q}_{I,J}$, then, the null vector space of A (i.e. the kernel) is written $Ker(A) = \{v \in \mathbb{Q}^J / Av = \mathbf{0}\}$.

All notations are quite classical except $R(n)$ for the range of integer from 1 to n, $\mathbb{Q}_{I,J}$ is the set of matrix with I rows and J columns (more often written $M_{I,J}(\mathbb{Q})$ and $\mathbb{U}$ to indicate normalization of vector and/or matrix.

2 Slide and jump algorithm

2.1 Key idea

Let quickly consider the following lemma:

Lemma 2.1. *if it is possible to solve* $\exists ?x \ / \ Ax = \mathbf{0},\ x > \mathbf{0}$ *when* $A \in \mathbb{Q}_{M,N}$ *has full rank, then it is possible to solve* $\exists ?y, \mathcal{A}y > \mathbf{0}$ *for any* $\mathcal{A} \in \mathbb{Q}_{I,J}$.

Proof. From $\mathcal{A} \in \mathbb{Q}_{I,J}$, it is sufficient for that to consider the matrix $A \in \mathbb{Q}_{I,J\times 3}$ formed by $\mathcal{A}$ concatenate with $-\mathcal{A}$ concatenate with $-\mathbf{I}$ (which has full rank due to the identity bloc). Solution can be extracted from $J \times 2$ first variables of derived problem, J last variable ensure positivity. This is more formally proven in appendix. □

With the ability to call a strong polynomial time subsolver dedicated to linear separability ($Ax > \mathbf{0}$), it is easy to get a small improvement from a not optimal admissible solution either for linear program or support vector margin.

Indeed, if w verifying $Aw \geq \mathbf{1}$ has not minimal norm, then it is possible to find u such that $\begin{pmatrix} A \\ -w^T \end{pmatrix} u > \mathbf{0}$ (this will be proven more formally in section 2.3). Now, it could take infinite time to reach the optimal solution by computing such u and updating $w = w + \varepsilon u$. These moves can be see as *jump* because its allow to exit a situation but its are not sufficient.

Inversely, the algorithm considers *sliding* moves: it considers the current most problematic constraints e.g. D such that $A_D w = \mathbf{1}$, and it solves the easy optimization problem derived by freezing this set i.e. it solves $\min_{w \ / \ AD=\mathbf{1}} w^T w$. This problem can be trivially solved because it is just a projection on a vectorial space (can be done for example with gram schmidt basis transform).

So, *sliding* moves will allow the algorithm to explore completely a particular combinatorial situation (D) and *jumping* moves will allow to exit this structure. Thus, the termination will be guarantee because D can not be observed again as it has been completely explored (this will be proven more formally in section 2.3).

The main key point of this short paper is that the *same* slide and jump idea can solve different problems, at least linear program and support vector margin.

Unfortunately, the algorithm may be exponential because there is a lot of possible situations D (of course exploration can not lead to visit all of them, but, this point is not enough investigated in this short paper). Still, this algorithm is also interesting because it is strongly polynomial for both these two problems under the *same* kind of assumptions linked with convex hull properties.

2.2 Pseudo code

For linear program, the slide and jump algorithm assumes the input linear program is $\min_{x\in\mathbb{Q}^N, \ / \ Ax\geq b} c^T x$ with $A \in \mathbb{U}_{M,N}$, $b \in \mathbb{Q}^M$, $c \in \mathbb{U}_N$, $Ac = \frac{3}{5}\mathbf{1}$, x_{start}

being a trivial admissible point e.g. $(1 + \frac{5}{3}\max_m b_m)c$, and, $c^T x$ being bounded by 0.

Yet, this does not restrict generality because all linear program can be derived under this form (see appendix).

Pseudo code for linear program is presented in algorithm 1.

Algorithm 1 Slide and jump for linear program

$d = \min_{m \in R(M)} A_m x - b_m$
$D = \{m \in R(M) \ / \ A_m x - b_m = d\}$
compute u the orthogonal projection of $-c$ on $Ker(A_D)$
if $cu < 0$, $A_D u = \mathbf{0}$ **then**
    $g = \min_{m \in R(M) \ / \ A_m u < 0} \frac{A_m x - b_m - d}{-A_m u}$
    $x = x + gu$
    GO TO 1
call subsolver $\exists ? v \ / \ \begin{pmatrix} A_D \\ -c^T \end{pmatrix} v > \mathbf{0}$
$z = x - \frac{5d}{3} c$
if v not exists **then**
    return z and the certificate
$v = \frac{1}{\min_{m \in D} A_m v} v$
$h = \min_{m \in R(M) / A_m v < 0} \frac{A_m z - b_m}{-A_m v}$
$x = z + \frac{h}{4} v$
GO TO 1

For support vector margin, the slide and jump algorithm assumes the input problem is $\min_{w \in \mathbb{Q}^N, \ / \ Aw \geq \mathbf{1}} w^T w$ with $A \in \mathbb{U}_{M,N}$.

Again this does not restrict generality because in fact the algorithm can solve any $\min_{w \in \mathbb{Q}^N, \ / \ Aw \geq b > \mathbf{0}} w^T w$ (so normalization of appendix is also possible). Yet, considering $\min_{w \in \mathbb{Q}^N, \ / \ Aw \geq \mathbf{1}} w^T w$ with $A \in \mathbb{U}_{M,N}$ is relevant from machine learning point of view (this assumption gives equivalent importance to all vectors) and allows to a simpler expression of the hypothesis leading to strong polynomial time.

Also, a single call to the subsolver allows to know if $Aw \geq \mathbf{1}$ admits a solution ($\exists w \ / \ Aw > \mathbf{0} \Leftrightarrow \exists w \ / \ Aw \geq \mathbf{1}$ because it is sufficient to scale the vector by the min of $A_m w$), so, algorithm can assume to start from an admissible w.

Pseudo code for support vector margin is presented in algorithm 2.

2.3 Termination

This subsection presents a proof that both these algorithms are well defined, terminate, and produce an exact optimal solution.

Algorithm 2 Slide and jump for support vector margin

1: $w = \frac{1}{\min\limits_{m \in R(M)} A_m w} w$
2: $D = \{m \in R(M) \ / \ A_m w = 1\}$
3: compute u be the orthogonal projection of $-w$ on $Ker(A_D)$
4: **if** $wu < 0$, $A_D u = \mathbf{0}$ **then**
5: **if** $A(w + u) \geq \mathbf{1}$ **then**
6: $w = w + u$
7: **else**
8: $g = \min\limits_{m \in R(M)/A_m u < 0} \frac{A_m w - 1}{-A_m u}$
9: $w = w + gu$
10: **GO TO 2**
11: call subsolver $\exists ? v \ / \begin{pmatrix} A_D \\ -w^T \end{pmatrix} v > \mathbf{0}$
12: **if** v not exists **then**
13: **return** w and the certificate
14: **if** $A(w - \frac{wv}{vv} v) \geq \mathbf{1}$ **then**
15: $w = w - \frac{wv}{vv} v$
16: **else**
17: $v = \frac{1}{\min\limits_{m \in D} A_m v} v$
18: $h = \min\limits_{m / A_m w < 0} \frac{A_m w - 1}{-A_m v}$
19: $w = w + \frac{h}{4} v$
20: **GO TO 1**

Lemma 2.2. *All steps of both algorithms are well defined from an admissible point.*

Proof. Both algorithms consider minimum on conditional subset of $R(M)$. It is required to ensure these subsets are not empty. Minimum on the complete set are not concerned as $R(M)$ is expected not to be empty. Projection of a vector into a vectorial space is well defined operation (a space at least contains $\mathbf{0}$). All other operations are just arithmetic or logic operations or call to the subsolver.

So problematic steps are 5, 13 for linear program and 8, 18 for support vector margin.

For linear program, by assumption cx is bounded i.e. $\forall x \in \mathbb{Q}^N$, $Ax \geq b \Rightarrow cx \geq 0$. So, if there is ω such that $c\omega < 0$ and $\forall m, A_m\omega \geq 0$, then, one could produce an unbounded admissible point $x + \lambda\omega$ as $A(x + \lambda\omega) \geq Ax \geq b$ and $c(x + \lambda\omega) \underset{\lambda\to\infty}{\to} -\infty$.

So, steps 5 and 13 (of linear program code) are well defined.

For support vector margin, when algorithm reaches step 8, it means $\exists m \in R(M)$ / $A_m(w+u) < 1$, independently $Aw \geq 1$, so, $A_m u \leq A_m(w+u) - 1 < 0$, so the subset considered in step 8 is not empty. The same idea can be used for step 18: $\exists m \in R(M)$ / $A_m(w + \frac{wv}{vv}v) < 1$ and $A_m w \geq 1$, so, $-\frac{wv}{vv}A_m v < 0$ and vw is strict negative so $A_m v < 0$.

So, steps 8 and 18 (of support vector margin code) are well defined. □

Lemma 2.3. *All loops of both algorithms keep the current point in the admissible space (even in the interior of admissible space for linear program version).*

Proof. For linear program, only steps 6, 9, 14 modify x. For support vector margin, this is in steps 1, 6, 8, 15, 19.

In step 6 of linear program code, let m such that $A_m u < 0$, seeing test 4 $m \notin D$, and so that $A_m x - b_m > d > 0$ by definition of d, D. So $g > 0$. So, $\frac{A_m x - b_m - d}{-A_m u} \geq g > 0$ Now, $\frac{A_m x - b_m - d}{-A_m u} A_m u \leq g A_m u$ (product by a negative). So, $A_m x + g A_m u - b_m \geq A_m x + \frac{A_m x - b_m - d}{-A_m u} A_m u - b_m = d > 0$.

So step 6 of linear program code keeps x in the interior of admissible space (trivially, if $A_m u \geq 0$, than $A_m(x + gu) - b_m \geq A_m x - b_m \geq d > 0$).

The same idea works for step 14 of linear program code: let m such that $A_m v < 0$, seeing test 11 and definition of v it implies that $m \notin D$, which implies that $A_m x - b_m > d$, and so that $A_m z - b_m > 0$ as $z = x - \frac{5d}{3}c$ and by assumption $A_m c = \frac{3}{5}$. So $h > 0$.

Again, $A_m z + \frac{1}{4}hA_m w - b_m \geq A_m z + \frac{1}{4}\frac{A_m z - b_m}{-A_m v}A_m v - b_m = \frac{3}{4}(A_m z - b_m) > 0$ Also, if $A_m v = 0$, $A_m(x + hv) - b_m = A_m x - b_m > 0$. And, if $A_m z - b_m = 0$, $A_m v > 0$ so $A_m(x + hv) - b_m = hA_m v > 0$.

Finally, for step 9, $Az = A(x - \frac{5d}{3}c) \geq b + d\mathbf{1} - \frac{5d}{3}\frac{3}{5}\mathbf{1} \geq b$ because $Ac = \frac{3}{5}\mathbf{1}$.

So, for linear program, if x verifies $Ax > b$, then, it will still verify this property after step 6 or 14, or it will verify $Ax \geq b$ on return (step 9).

Now, for support vector margin, step 1 trivially keeps the point in the border of the admissible space by construction. Steps 6 and 15 are safe when taking place.

In step 8, let m such that $A_m u < 0$, seeing test 4 $m \notin D$, and so that $A_m w > 1$. So, $g > 0$. Then $A_m(w + gu) \geq A_m w + \frac{A_m w - 1}{-A_m u} A_m u = A_m w + 1 - A_m w = 1$.

In step 19, let m such that $A_m v < 0$, seeing definition of v, $m \notin D$, and so that $A_m w > 1$. So, $h > 0$. Then $A_m(w + \frac{h}{4}v) - 1 \geq A_m w + \frac{h}{4}\frac{A_m w - 1}{-A_m v} A_m v - 1 = \frac{3}{4}(A_m w - 1) > 0$. □

Lemma 2.4. *If algorithms terminate, its produce optimal point.*

Proof. Let assume the algorithm returns non optimal admissible point z, w while optimal solution were z^* and w^* ($cz > cz^*$ and $(w^*)^T(w^*) < w^T w$).

For linear program, let consider $\phi = z^* - z + \frac{c^T(z - z^*)}{2}c$. $A_D\phi = A_D(z^* - z + \frac{c^T(z-z^*)}{2}c) = A_D(z^* + \frac{c^T(z-z^*)}{2}c) \geq \frac{c^T(z-z^*)}{2}A_D c > \mathbf{0}$ and $c(\phi - z) = -\frac{c^T(z-z^*)}{2} < 0$. So, test step 10 should have been false because ϕ is a possibility for v. So, algorithm should not have returned on z.

For support vector margin, let $\psi = (\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})w^* - w$ (currently, ψ is not in $\mathbb{Q}^N$ but only matter the fact that it exists).

Then, $A_D\psi = (\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})A_D w^* - A_D w = (\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})A_D w^* - \mathbf{1} \geq (\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})\mathbf{1} - \mathbf{1} \geq \frac{1}{2}(\sqrt{\frac{w^T w}{(w^*)^T(w^*)}} - 1)\mathbf{1} > \mathbf{0}$ (because w^* is admissible and $(w^*)^T(w^*) < w^T w$). Independently, norm of $(\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})w^*$ is strictly less than $\sqrt{\frac{w^T w}{(w^*)^T(w^*)}(w^*)^T(w^*)} = \sqrt{w^T w}$. So, $w^T((\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})w^*) < \sqrt{w^T w}\sqrt{w^T w} = w^T w^T$ (because $\alpha^T\beta \leq \sqrt{\alpha^T\alpha} \leq \sqrt{\beta^T\beta}$). So $w\psi = (\sqrt{\frac{w^T w}{4(w^*)^T(w^*)}} + \frac{1}{2})w^T w^* - w^T w < w^T w - w^T w = 0$.

So, test step 12 should have been false because ψ is a possibility for v. So, algorithm should not have returned on w. □

Lemma 2.5. *Both algorithms can not loop more then $M + 1$ times without calling the subsolver.*

Proof. In step 6 of linear program code, let consider an index $k \notin D$ such that $\frac{A_k x - b_k - d}{-A_k u} = g$. So, $\frac{A_k x - b_k - d}{-A_k u} A_k u = g A_k u$. And, so, $A_k(x + gv) - b_k = A_k x + \frac{A_k x - b_k - d}{-A_k v} A_k v - b_k = d$. After step 6, if $\frac{A_k x - b_k - d}{-A_k u} = g$, then k enters in D.

So in linear program code, D strictly increases each time the algorithm reaches step 6. But, D is bounded by $R(M)$.

For support vector margin, if program reaches step 6, w will become $w + u$ with u the orthogonal projection of $-w$ on $Ker(A_D)$. But, the projection of u is itself (by definition of a projection). So, the projection of $-(w + u)$ on $Ker(A_D)$ is $u + (-u) = \mathbf{0}$. So, if the program reaches step 6, next loop does not enter test step 4 and algorithm will call subsolver on step 11.

Independently, in step 9, let consider an index $k \notin D$ such that $\frac{A_k x - 1}{-A_k u} = g$. $A_k(w + gu) = A_k w + \frac{A_k w - 1}{-A_k u} A_k u = A_k w + 1 - A_k w = 1$. So, k enters in D in next loop.

Like for linear program, D strictly increases each time the algorithm reaches step 9, but, D is bounded by $R(M)$ (and if it reaches step 6, it directly goes step 11 after the loop). □

Lemma 2.6. *A value of set D can not be observed during the algorithm after a call to the subsolver on this value.*

Proof. First, all moves strictly decreases $c^T z$ or $w^T w$ because by construction $u^T c < 0$ and $v^T c < 0$ or $u^T w < 0$ and $v^T w < 0$ and $g, h > 0$.

Independently, when reaching step 8 in linear program code or step 11 in support vector margin one, it means that projection of $-c$ or $-w$ on $Ker(A_D)$ does not lead to u such that $A_D u = 0$ and $c^T u < 0$ or $w^T u < 0$.

Now, let assume to observe D again after having called subsolver.

For linear program, let consider $z_2 - z_1$ with z_2 being $z = x - \frac{5}{3}c$ when observing D again, and, z_1 be z when observing D just before calling the subsolver. As all moves strictly decreases $c^T z$, it means $c^T(z_2 - z_1) < 0$, but, by definition of D, $A_D z_2 = A_D z_1 = \mathbf{0}$ so $A_D(z_2 - z_1) = \mathbf{0}$. So, algorithm should not have passed the test step 4 when meeting z_1, because $z_2 - z_1$ should have been considered. This is a contradiction.

For support vector margin code, let consider $w_2 - w_1$. First, $A_D(w_2 - w_1) = 0$. Then, $w_1^T(w_2 - w_1) = w_1^T w_2 - w^1 w_1 \leq \sqrt{w_1^T w_1^T}\sqrt{w_2^T w_2^T} - w^1 w_1 < \sqrt{w_1^T w_1^T}\sqrt{w_1^T w_1^T} - w_1^T w_1^T = 0$. So, again, algorithm should not have passed the test step 4 when meeting w_1, because $w_2 - w_1$ should have been considered. □

From all previous lemmas, observing that D is bounded (so looping without call to subsolver is impossible - lemma 2.5), and that, number of subset D from $R(M)$ is finite (so number of call is bounded - lemma 2.6), it holds that:

Theorem 2.7. *The slide and jump algorithms solve both linear program and support vector margin.*

3 Complexity analysis

3.1 Special cases with strongly polynomial time termination

Termination is proving observing that a call of subsolver on a set D will forbid to see D again. Unfortunately, there is an exponential number of possible value for D. So, slide and jump algorithms may be exponential. Even more, if algorithms were not exponential, it would be because the dynamic of the exploration prevents for exploring too much subsets D. But, nothing in the structure of the algorithms prevents an exponential behaviour.

Yet, in some situation, the exploration of a set D allows to be sure that at least some constraint will never be seen again during all the algorithm. And, this is very useful as there is only M constraints.

So, in this section, slide and jump algorithm is shown to be **strong polynomial** for specific family of instances of linear program and support vector margin characterized by convex hull property.

First, let introduce some notations related to convex hull:

$$\forall \mathcal{A} \in \mathbb{Q}_{I,J}, \text{ let write:}$$
$$Conv(\mathcal{A}) = \{\mathcal{A}^T\pi \ / \ \pi \in \mathbb{Q}^I, \pi \geq \mathbf{0} \wedge \mathbf{1}^T\pi = 1\}$$
$$Conv_+(\mathcal{A}) = \{\lambda\theta \ / \ \theta \in Conv(\mathcal{A}), \lambda \in \mathbb{Q}, \lambda \geq 1\}$$

$Conv(\mathcal{A})$ is the convex hull of rows of $\mathcal{A}$ i.e. the set of linear combination with scalar coefficients being in the simplex of $\mathbb{Q}^M$ of rows of $\mathcal{A}$ (after transposition). And $Conv_+(\mathcal{A})$ is the set of vectors x that can be scaled by a number $\frac{1}{\lambda}$ with λ higher than 1 such that $\frac{1}{\lambda}x \in Conv(\mathcal{A})$.

One step for linking convex hull and the complexity of slide and jump algorithm is to observe that:

Lemma 3.1. $a \in Conv(\mathcal{A}) \Rightarrow \nexists \omega \in \mathbb{Q}^N, \ / \ \mathbf{0} \leq a^T\omega\mathbf{1} < A\omega$

Proof. if both ω exists and $a = A^T\pi$, then, $\omega^T a = \omega^T A^T\pi = (A\omega)^T\pi > (\omega^T a\mathbf{1}^T)\pi = (\omega^T a)(\mathbf{1}^T\pi) = \omega^T a$ i.e. $\omega^T a > a^T\omega$. □

And, even,

Lemma 3.2. $a \in Conv_+(\mathcal{A}) \Rightarrow \nexists \omega \in \mathbb{Q}^N, \ / \ \mathbf{0} \leq a^T\omega\mathbf{1} < A\omega$

Proof. $a \in Conv_+(\mathcal{A})$ means that $\exists\lambda \geq 1$ such that $\frac{1}{\lambda}a \in Conv(\mathcal{A})$. For previous lemma, there is not ω such that $\mathbf{0} < \frac{1}{\lambda}a^T\omega\mathbf{1} < A\omega$. Now, if there was ω such that $\mathbf{0} < a^T\omega\mathbf{1} < A\omega$, then obviously, $\mathbf{0} < \frac{1}{\lambda}a^T\omega\mathbf{1} < a^T\omega\mathbf{1} < A\omega$ as $\lambda \geq 1$. □

The other step to connect convex hull and the complexity of slide and jump algorithm is to observe that $\exists\omega \in \mathbb{Q}^N, \ / \ \mathbf{0} \leq a^T\omega\mathbf{1} < A\omega$ is characteristic of being into D. So, if some constraints are in the convex hull of some other, then its can never enter into D. So let state the main theorem of this paper:

Theorem 3.3. *Let D^* be the set of support vectors on termination, if for all $m \in R(M)$, $A_m \in Conv_+(A_{D^*})$, then, the algorithm terminates with two calls to the subsolver and at most $M+1$ sliding moves: one time to initialize w, and, one time to check that the solution is optimal, and at most $M+1$ sliding moves between both calls (all moves beeing strongly polynomial).*

Proof. If $k \notin D^*$ is at some point into $D \neq D^*$, then let consider $\theta = w - A_k^T$.

$A_k\theta = 1 - 1 = 0$ ($A_k w = 1$ because $k \in D$, and, $A_k^T A_k = 1$ by assumption).

But, $\forall m \in D^*$, $A_m\theta \geq 1 - A_m A_k^T > 1 - 1 = 0$ except if $A_m A_k^T = 1$ which is impossible as $k \notin D^*$.

This is a direct contradiction with the lemma 3.2.

So none of the $k \notin D^*$ can be in D, so the algorithm will start with a subset of D^* and performs at most $M+1$ sliding moves to collect all the set D^*, and, terminates after calling the subsolver to get a certificate. □

For linear program code, let consider $A_D z = b$ on which the subsolver is called. Let assume $\exists k \in D$ such that $A_k \in Conv_+(\begin{smallmatrix} A_{D\setminus\{k\}} \\ -c^T \end{smallmatrix})$.

Let suppose k is see on an other set D after having called the subsolver. Let project x on constraint k i.e. $\theta = x - dA_k$.

Now, $A_k\theta = 0$ by definition.

And, $-c^T\theta < 0$ because x is strictly bellow z (bellow from c point of view) and $c^T A_k > 0$.

And, $A_m\theta \geq d - dA_m^T A_k > 0$.

This is a direct contradiction with the lemma 3.2.

So, it holds:

Theorem 3.4. *If for any, $A_D z = b$ on which the subsolver is called, there is $k \in D$ such that $A_k \in Conv_+(\begin{smallmatrix} A_{D\setminus\{k\}} \\ -c^T \end{smallmatrix})$, then k is never observed in any ulterior D, and so, under this assumption, the algorithm for linear program terminates after at most M calls to the subsolver i.e. after at most M jumping moves and M^2 sliding moves (all beeing strongly polynomial).*

Remark: For support vector margin, this weaker version of strong polynomial time termination should also exist.

3.2 Discussion

So, algorithms always terminate. And in addition, under very strong hypothesis (e.g. that the convex hull of support vectors contains all the vectors scaled to have 1 scalar product to w^*), both algorithms are strongly polynomial.

Now, the interesting question is: how much these strong hypothesis are *realistic* or *relevant* ? In other words, the question is: is there argument that average complexity or smoothed complexity may be low because a lot of instances are close to verify these hypothesis or not ?

For support vector margin, algorithm considers $\min_{w \ / \ Aw \geq \mathbf{1}} w^T w$. The convex hull of A_m^T such that $A_m w^* = 1$ is in the hyper plane $H = \{u \ / \ u^T w^* = 1\}$. Now, all $\frac{1}{A_k v} A_k$ are in H and more precisely in $H \cap B$ with $B = \{u \ / \ u^T u \leq 1\}$ and A_i such that $A_i v = 1$ are border point of $H \cap B$.

So, loosely speaking, $H \cap B$ is the excircle of the convex hull of support vectors. So, for support vectors, the situation is a set of vectors D^*, from which one can consider the excircle of the corresponding convex hull. And, the algorithm will be ultra fast, if all other vectors (after scaling) are in the convex hull (knowing that there are in the excircle). Inversely, annoying vectors must be strictly between the convex hull and the corresponding excircle.

So, basically, the *relevancy* of the assumption which makes the algorithm strongly polynomial, is linked to the ratio of the volume of a convex hull by the volume of the corresponding excircle. For random vectors, this is a well studied question e.g. [7]. So, the number of instance verifying these assumptions may not be too low (notice that this is always verified for $N \leq 2$).

The same considerations are true for linear program. After reaching a point z such that all constraints can not be improved all equally, one can consider $\min_{\omega, A_D\omega \geq \mathbf{1}, c^T\omega=0} \omega^T\omega$. In not degenerated case, D contains exactly N vectors, so ω will fit $N-1$ from these N. Then the question is: is the last one (the one such that $A_k\omega > 1$) in the convex hull of the others (after scaling such that $A_k\omega = 1$) ? If the answer is true, it means, k will never enter again in D because, it is not possible to be closer to k than to $D\backslash\{k\}$. So there is a strong constraint forgetting. If the answer is false, then it is an annoying vector.

So, again, an annoying vector should be strictly between the convex hull (of the others) and the corresponding excircle. So, the number of instance verifying these assumptions may again not be too low.

Discussion

This short paper presents an algorithm based on the idea of greedy optimisation with frozen combinatorial structure, and, linear feasibility queries which allows to refine the combinatorial structure. To give a metaphor, greedy moves slides on linear constraints, and linear feasibility allows to jump on new ones.

The first interest of this *slide and jump* idea is it ability to solve both linear program and support vector margin (both algorithms are almost identical).

And, the second is that algorithms are strongly polynomial for both problems under a same convex hull assumption. Thus, this short paper presents a link between linear feasibility, linear program, support vector margin and geometric complexity of convex hull.

References

[1] Fernando Affentranger. The expected volume of a random polytope in a ball. *Journal of Microscopy*, 151(3):277–287, 1988.

[2] Xavier Allamigeon, Pascal Benchimol, Stéphane Gaubert, and Michael Joswig. Log-barrier interior point methods are not strongly polynomial. *SIAM Journal on Applied Algebra and Geometry*, 2(1):140–178, 2018.

[3] Olivier Chapelle. Training a support vector machine in the primal. *Neural computation*, 19(5):1155–1178, 2007.

[4] Sergei Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Mathematical programming*, 134(2):533–570, 2012.

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

[6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[7] A Giannopoulos and A Tsolomitis. Volume radius of a random polytope in a convex body. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 134, pages 13–21. Cambridge University Press, 2003.

[8] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[9] Christoph Helmberg, Franz Rendl, Robert J Vanderbei, and Henry Wolkowicz. An interior-point method for semidefinite programming. *SIAM Journal on Optimization*, 6(2):342–361, 1996.

[10] Didier Henrion, Simone Naldi, and Mohab Safey El Din. Exact algorithms for semidefinite programs with degenerate feasible set. *arXiv preprint arXiv:1802.02834*, 2018.

[11] Dorit S Hochbaum and Joseph Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, 1994.

[12] Don Hush, Patrick Kelly, Clint Scovel, and Ingo Steinwart. Qp algorithms with guaranteed accuracy and run time for support vector machines. *Journal of Machine Learning Research*, 7(May):733–769, 2006.

[13] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[14] S Sathiya Keerthi and Dennis DeCoste. A modified finite newton method for fast solution of large scale linear svms. *Journal of Machine Learning Research*, 6(Mar):341–361, 2005.

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

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Josef S Müller. Approximation of a ball by random polytopes. *Journal of Approximation Theory*, 63(2):198–209, 1990.

[18] Yurii Nesterov and Arkadii Nemirovskii. *Interior-point polynomial algorithms in convex programming*, volume 13. Siam, 1994.

[19] Pablo Pérez-Lantero. Area and perimeter of the convex hull of stochastic points. *The Computer Journal*, 59(8):1144–1154, 2016.

[20] Ian Post and Yinyu Ye. The simplex method is strongly polynomial for deterministic markov decision processes. *Mathematics of Operations Research*, 40(4):859–868, 2015.

[21] Eva Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

Appendix

Linear feasibility and linear separability

[5] presents an algorithm to solve in strongly polynomial time the following problem: $\exists ?x \in \mathbb{Q}^N / Ax = \mathbf{0}, x > \mathbf{0}$ under the assumption that $A \in \mathbb{Q}_{M,N}$ has a rank of M.

Let $\mathcal{A}$ a matrix without any assumption. Let consider the matrix $A = \begin{pmatrix} \mathcal{A} & -\mathcal{A} & -\mathbf{I} \end{pmatrix}$ formed with $\mathcal{A}$ concat with $-\mathcal{A}$ concat with $-\mathbf{I}$ the opposite of identity matrix.

First, this matrix has full rank M due to the identity block.

Then, applying the Chubanov algorithm (or any linear feasibility solver) to this matrix A will lead (if a solution exists) to x_1, x_2, x_3 such that

$$\begin{pmatrix} \mathcal{A} & -\mathcal{A} & -\mathbf{I} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \mathbf{0}$$

and $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} > \mathbf{0}$ So, let $x = x_1 - x_2$, it holds that $\mathcal{A}x = \mathbf{I}x_3 = x_3 > \mathbf{0}$. And, if no solution exists, then Chubanov algorithm will provide a certificate.

Chubanov algorithm can be applied on a derived problem to solve linear separability $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x > \mathbf{0}$ with $\mathcal{A} \in \mathbb{Q}_{M,N}$
(under no assumption on $\mathcal{A}$, especially on the rank).

Normalizing linear program

If the linear program given as input is $\min_{Ax \geq b} cx$ and verifies $A \in \mathbb{U}_{M,N}$, $b \in \mathbb{Q}^M$, $c \in \mathbb{U}_N$ and $Ac = \gamma \mathbf{1}$ with $\gamma > 0$, and, cx being bounded by 0, then the offered algorithm of section 2 can be directly used.

Otherwise, the linear program has to be normalized with the following scheme:

1. If the linear program is as an optimisation problem (e.g. $\max_{Ax \leq b, x \geq \mathbf{0}} cx$), it should first be converted into a inequality system $A'x \geq b'$. This could be done by combining primal and dual.
2. After that (or directly is input was an inequality system), an other normalisation is performed to reach required property (here with $\gamma = \frac{3}{5}$)
3. the important point is that from any linear program, pre processing can form an equivalent linear program meeting these requirements

Primal dual

The conversion of an linear program to be optimized into a linear inequality system is quite classical. A brief recall is provided bellow.

Let assume original goal is to solve $\max_{A_{raw}x \leq b_{raw}, x \geq \mathbf{0}} c_{raw}x$. It is well known that the dual problem is $\min_{A_{raw}^T y \geq c_{raw}, y \geq \mathbf{0}} b_{raw}y$. Now, the primal dual is formed by combining all constraints: $A_{raw}x \leq b_{raw}$, and, $x \geq \mathbf{0}$, and, $A_{raw}^T y \geq c_{raw}$, and $c_{raw}x = b_{raw}y$, and finally, $y \geq \mathbf{0}$.

So, the problem $\max_{A_{raw}x \leq b_{raw}, x \geq \mathbf{0}} c_{raw}x$ can be folded into $A_{big}x_{big} \geq b_{big}$ with

$$A_{big} = \begin{pmatrix} -A_{raw} & 0 \\ I & 0 \\ 0 & A_{raw}^T \\ 0 & I \\ c_{raw} & -b_{raw} \\ -c_{raw} & b_{raw} \end{pmatrix} \text{ and } b_{big} = \begin{pmatrix} -b_{raw} \\ 0 \\ c_{raw} \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Normalized primal dual error minimization

This normalisation step takes a linear program $\Gamma\chi \geq \beta$ as input, and, produces an equivalent linear program $\min_{Ax \geq b} cx$ with $A \in \mathbb{U}_{M,N}$, $b \in \mathbb{Q}^M$, $c \in \mathbb{U}_N$, and, $Ac = \frac{3}{5}\mathbf{1}$, and, cx being bounded by 0.

It is sufficient to consider:

$$A = \begin{pmatrix} \frac{4}{5(\frac{\Gamma_1\Gamma_1}{2}+1)}\Gamma_1 & \frac{4}{5(\frac{\Gamma_1\Gamma_1}{2}+1)}\frac{\Gamma_1\Gamma_1}{2} & \frac{4}{5(\frac{\Gamma_1\Gamma_1}{2}+1)} & \frac{3}{5} \\ \cdots & \cdots & \cdots & \\ \frac{4}{5(\frac{\Gamma_M\Gamma_M}{2}+1)}\Gamma_M & \frac{4}{5(\frac{\Gamma_M\Gamma_M}{2}+1)}\frac{\Gamma_M\Gamma_M}{2} & \frac{4}{5(\frac{\Gamma_M\Gamma_M}{2}+1)} & \frac{3}{5} \\ \mathbf{0} & \frac{4}{5} & 0 & \frac{3}{5} \\ \mathbf{0} & -\frac{4}{5} & 0 & \frac{3}{5} \\ \mathbf{0} & 0 & \frac{4}{5} & \frac{3}{5} \\ \mathbf{0} & 0 & -\frac{4}{5} & \frac{3}{5} \end{pmatrix}$$

and

$$b = \begin{pmatrix} \frac{4}{5(\frac{\Gamma_1\Gamma_1}{2}+1)}\beta_1 \\ \cdots \\ \frac{4}{5(\frac{\Gamma_M\Gamma_M}{2}+1)}\beta_M \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad , \quad c = \begin{pmatrix} 0 \\ \cdots \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

First, the produced linear program is in the desired form: $\min_{Ax \geq b} cx$ with $A \in \mathbb{U}_{J,I}$, $b \in \mathbb{Q}^M$, $c \in \mathbb{U}_N$, and, $Ac = \frac{3}{5}\mathbf{1}$.

Trivially, $Ac = \frac{3}{5}\mathbf{1}$ by construction, and, all rows of A are normalized either directly because $(\frac{4}{5})^2 + (\frac{3}{5})^2 = 1$, or, because of that, and the fact that, $(\frac{1}{\frac{\Gamma_m\Gamma_m}{2}+1})^2\Gamma_m\Gamma_m + (\frac{1}{\frac{\Gamma_m\Gamma_m}{2}+1})^2\frac{(\Gamma_m\Gamma_m)^2}{4} + (\frac{1}{\frac{\Gamma_m\Gamma_m}{2}+1})^2$ is $(\frac{1}{\frac{\Gamma_m\Gamma_m}{2}+1})^2 \times (\Gamma_m\Gamma_m + \frac{(\Gamma_m\Gamma_m)^2}{4} + 1)$ which is $(\frac{1}{\frac{\Gamma_m\Gamma_m}{2}+1})^2 \times (\frac{\Gamma_m\Gamma_m}{2} + 1)^2$ which is 1 !

Then, the 4 last constraint prevent x_{N+3} to be negative so cx is well bounded by 0. Indeed, if $x_{N+2} + x_{N+3} \geq 0$ and $-x_{N+2} + x_{N+3} \geq 0$, then $x_{N+3} \geq 0$), and, when $x_{N+3} = 0$ these constraints force $x_{N+1} = x_{N+2} = 0$ because the three constraints $x_{N+2} + x_{N+3} \geq 0$, $-x_{N+2} + x_{N+3} \geq 0$, and $x_{N+3} = 0$ can be reduced to $x_{N+2} \geq 0$ and $-x_{N+2} \geq 0$ which force $x_{N+2} = 0$.

Now, the goal is to minimize $cx = x_{N+3}$. So, either the minimum is $x_{N+3} = 0$ or either there is no such solution. In the case $Ax \geq b, x_{N+3} = 0$, it holds $x_{N+1} = x_{N+2} = x_{N+3} = 0$, and, $x_1, ..., x_N = \chi$ with $\frac{4}{5(\frac{\Gamma_m\Gamma_m}{2}+1)}\Gamma_m\chi \geq \frac{4}{5(\frac{\Gamma_m\Gamma_m}{2}+1)}\beta_m$. But this last inequality can be reduced to $\Gamma_m\chi \geq \beta_m$. So, if the solution of the derived linear program is x with $Ax \geq b, x_{N+3} = 0$, then $x_1, ..., x_N = \chi$ is a solution of the original set of inequality.

And, inversely, if there is a solution χ, then, $x = \chi, 0, 0, 0$ is a solution of the optimisation problem (because x_{N+3} is bounded by 0).

So, this derived linear program is equivalent to the inequality set.

For any linear program, it is possible to create a derived form meeting the requirement of the offered algorithm.
All this normalization is entirely done in $\mathbb{Q}$ i.e. no square root are needed.