Bisection based algorithm for linear program.

Adrien Chan-Hon-Tong

Bisection based algorithm for linear program.

Adrien CHAN-HON-TONG
ONERA - université paris sud

January 2019

Abstract

Thank to Chubanov algorithm, there is today a strongly polynomial time algorithm to solve linear feasibility problem. In this paper, we offer an algorithm for generic linear program based on bisection space using Chubanov algorithm as routine. The complexity of the algorithm is not established but is linked to geometric statements.

1 Introduction

Solving linear program can be done in polynomial time since ellipsoid method (e.g. [3]) and interior point (e.g. [4]).

But there is no known algorithm to solve generic linear program in strongly polynomial time. Indeed, [1] proves that one of the most important current interior point algorithm is not strongly polynomial. Yet, for some family of linear program, specific algorithm provide strong polynomial complexity including combinatorial linear program [5], linear program with less than 2 variables per constraint, linear program with binary solution...

In this paper, I am especially interested by [2] which provides a strongly polynomial time algorithm for linear feasibility problem.

Indeed, I offer an algorithm using linear feasibility as routine to solve generic linear program. I do NOT claim that the offered algorithm is strongly polynomial. Yet, I link some geometric statement to key possible feature of the offered algorithm.

2 Some lemma

2.1 Required routine

First of all, [2] allows 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 $A \in \mathcal{M}_{M,N}(\mathbb{Q})$ has a rank of M.

A minor but required lemma is that with a simple trick, this algorithm can be used to solve (under the assumption that some solution exists):

$$\exists ?x \in \mathbb{Q}^N / \mathcal{A}x > \mathbf{0}$$

with any $\mathcal{A} \in \mathcal{M}_{M,N}(\mathbb{Q})$.

Indeed, it is sufficient to consider the matrix $A = \begin{pmatrix} \mathcal{A} & -\mathcal{A} & -I \end{pmatrix}$ formed with $\mathcal{A}$ concat with $-\mathcal{A}$ concat with $-I$ the identity matrix. Applying the Chubanov algorithm to this matrix A will lead to x_1, x_2, x_3 such that

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

and $\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} > \mathbf{0}$ So, $x = x_1 - x_2$ and $\mathcal{A}x = Ix_3 = x_3 > \mathbf{0}$.

Let note that the rank is obviously M as there is a identity bloc. Let also note that [2] with this trick solves in strong polynomial time the machine learning problem of linear separability (related to support vector machine SVM but without the property of maximizing the minimal distance to this hyper plane). To my knowledge this has not be that stressed yet... Finally, let not that $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x > \mathbf{0}$ is equivalent of $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x \geq 1$: solution of the second is solution of the first, and solution of the first can be scaled to become one of the second.

In the offered algorithm, this is this routine i.e. solving $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x > \mathbf{0}$ that why be required.

It will never be called without be sure that a solution exist.

2.2 Pre processing

2.2.1 Required pre processing

The offered algorithm can be used to solve all possible king of linear program: $\min_{Ax=b, x \geq \mathbf{0}} cx$ or $\max_{Ax=b, x \geq \mathbf{0}} cx$ or $\min_{Ax \geq b} cx$ or $\max_{Ax \geq b} cx$ or $\exists ?x \in \mathbb{Q}^N / Ax \geq b$ or ...

But, this has to be done this way:

1. a derived linear program has to be generated from the raw targeted linear program
2. this new linear program will have the required properties for using the offered algorithm (whatever the raw input is)
3. then solution of the input linear program has to be extracted from solution of the derived one (providing certificate for unbounded or empty linear program)

Indeed, I will give systematic way to reach such derived linear program with required properties. In one sentence, it needs to combine both primal dual twice again. But, simpler transformation may be sufficient depending of the input linear program.

Anyway, the requirement is to manage to reach a derived linear program (whose solution contains solution of the raw one) $\min_{A_{final}x_{final} \geq b_{final}} c_{final}x_{final}$ which meets the following assumptions:

- $A_{final}c_{final} > \mathbf{0}$
  - it means that from any point x belonging to the admissible space, moving along c increase distance to all constraints
  - it implies that there is a non empty admissible space and a trivial solution (x = λc for large λ)
- there is an optimal solution and x_{final} is optimal iif $A_{final}x_{final} \geq b_{final}$ and $c_{final}x_{final} = 0$
- for any two normalized rows α, β of A_{final}, $\alpha c_{final} \neq \beta c_{final}$, it implies that moving along c modify the set of the most close constraint planes.

2.2.2 Example of implementation

The generation of a derived linear program from a standard $\max_{Ax \leq b, x \geq \mathbf{0}} cx$ is a combination of classical trick but this combination is rarely done due to the specificity of the requirement of this paper.

Let assume original goal is to solve $\max_{Ax \leq b, x \geq \mathbf{0}} cx$. It is well known that the dual problem is $\min_{A^T y \geq c, y \geq \mathbf{0}} by$. Now, the primal dual is formed by combining both $Ax \leq b, A^T y \geq c, cx = by$. This problem can be folded into a A_{big} matrix and a b_{big} as $A_{big}x_{big} \geq b_{big}$ (by the way the pre processing can be started here is the raw problem comes like $\exists ? x \in \mathbb{Q}^N / Ax \geq b$).

Now, this $A_{big}x_{big} \geq b_{big}$ can have no solution (it corresponds to an original problem unbounded and/or unsolvable). So, let normalize rows of the matrix, and, add some variable $\min_{A_{big}x_{big} + zg \geq b_{big}, z \geq 0} z$, with z being just a scalar and g the vector with $g_i = i$.

This last problem verifies almost all requirement except that the optimal value of z is not known. But, if $A_{big}x_{big} \geq b_{big}$ would have been a solution, this optimal value would have been 0.

So, this process is done ones again (this is not classical) leading to

$$\min_{A_{double}x_{double} + z_{double}g \geq b_{double}, z_{double} \geq 0} z_{double}$$

This derived linear program verifies the requirement. And, from solution of this linear program one can solve the original one.

3 Algorithm

3.1 Key points

Before introducing the algorithm, I present here some key points:

- The algorithm works on the interior of the admissible space like interior point method.
- But this algorithm is almost purely combinatorial: numerical values do not directly influence move of the current point.
- Distance between current point and constraint is computed and *closest* constraints push the point away (with constraint to not increase the cost function).
- Then, the idea is to try to update the less possible the set of the closest constraints. In other words, one can see rows of A, b represent planes in the space. If the distance between x and the closest plane is d than the idea is to move in a way that both d does not decrease and cx decreases (at least one strictly). Eventually, there will be 2 planes at distance d (because the move can not be infinite). But, the idea will still be to move without increasing the distance to any of two, then three and so one.
- Such keep-D move can always be trivially computed, but, such move may not exists ($Ax \geq 1$ could have a solution while $Ax = 1$ no). When no solution exists, the routine based on Chubanov algorithm is called leading to a reboot of the set.
- The algorithm terminates because the set of closest constraint planes can not be twice the same set
- Yet, in preliminary empirical results in 4D, a reboot of the plane set is associated with the definitive exclusion of at least 1 plane. This statement is probably wrong (except for 2D, 3D and simple cases) - but this statement is related to the complexity of this algorithm, and, in particular with an hypothetical strong polynomial time property.

To keep standard notation, N is the number of variables and M the number of constraint (i.e. rows of matrix A).

Transposition is omitted in scalar product: if p, q are 2 vectors pq corresponds to $p^T q = \sum_n p_n q_n$.

Constraints, rows of matrix A and planes will be 3 ways to speak about the same objets.

For simplicity, algorithm is described like if rows of the matrix and cost vector was normalized. Of course, this is not an acceptable assumption as normalization is not possible in $\mathbb{Q}$. But algorithm can be modified to remove this need for normalization.

3.2 Pseudo code

The structure of the algorithm is described by the following pseudo code (in the normalized case).

1. compute $d = \min_m A_m x - b_m$ (the minimal distance to planes)
2. compute D the set of planes at distance d
3. check trivial termination
   - if x is a solution ($Ax \geq b$, $cx = 0$) return x
   - if the plane $cx = 0$ is at distance d return $x - dc$
4. look[1] for a vector v which maintains D, decreases cx, increase d ; 2 simple options are:
   - the projection of $-c$ on $Ker(A_D)$ e.g. $cv < 0$, $A_D v = \mathbf{0}$
   - the projection of A_D on the ker of pairwise difference plus c e.g. $cv = 0$, $A_D v = \mathbf{1}$
5. if a vector $v \neq 0$ is found
   (a) compute $\alpha = A_m v$ for any $m \in D$ (common value)
   (b) compute[2] λ: $\min_{m \notin D} d + \lambda\alpha + b_m - A_m x - \lambda A_m v$
   (c) $x \leftarrow x + \lambda v$ and GO TO 1
6. compute[3] H a basis of all vectors h such that $\exists \mu / A_D h = \mu \mathbf{1}$
7. compute[4] w the projection of $-c$ on H e.g. $cw < 0$, $A_D w = -\mathbf{1}$
8. if $w = 0$
   (a) initialize $\mathcal{D} = \emptyset$
   (b) for all i in D (decreasingly sorted by the value of $A_i c$)
   add i to $\mathcal{D}$ iff doing so leads to a non zero w
   (c) compute ω the value of w if D was $\mathcal{D}$
   (d) find a $\lambda \in]0, d[$ such that the D set of $x + \lambda\omega$ is $\mathcal{D}$
   (e) GO TO 1 // see discussion visualize 8.a - 8.d as $x \leftarrow x - dc$
9. compute $\beta = A_m w$ for any $m \in D$ (common value)
10. compute z and λ such that $A_D z = 0$ and $z = \lambda w + x$
11. compute ρ: $\min_{m \notin D} \max(0,\ d + \rho\beta + b_m - A_m x - \rho A_m w)$
12. if $\rho \in]0, \lambda[$, $x \leftarrow x + \rho w$, GO[5] TO 1
13. compute $S = \{m / A_m z = b_m\}$ the set of planes saturated by z
14. call Chubanov routine to find θ such that $\begin{pmatrix} A_S \\ -c \end{pmatrix} \theta > \mathbf{0}$
15. find δ such that $A(z + \delta\theta) > b$
16. $x \leftarrow z + \delta\theta$ and GO TO 1

[1] notice that both these computations are just projection of a vector on a sub space so it is just a application of Gram-Schmidt

[2] here moving along v either decreases the cost of x - which is obviously bounded as $cx \geq 0$ - or either increases distance to closest planes which means that $c(x - dc)$ decreases which is bounded for the same reason. So this move can not be done for ever - necessarily some plane with meet the ball of center x and radius d - it will be added to D

[3] this is a generalisation of the bisectors from pair to set - the goal is to keep D

[4] the situation is very different from step 4 because here moving along w could lead to break constraint

[5] Moves from step 4 increases the distance to the border - these are safe moves. Here, the move is not safe as it could have ended in z - yet an other constraint was here making it acceptable

4 Properties of the algorithm

4.1 Termination

From an inner point, the algorithm generates a set of inner point plus one optimal point.

First, all moves are designed to never go outside the interior of the admissible space - except to reach an optimal point. Indeed, step 5 increases satisfaction of constraint. Step 8 takes care to make a move smaller than the radius of the ball centred on the point tangent to some constraint. Step 12 can be done only if the move maintains the point in the interior. Finally, step 16 starts from a corner and uses θ to increases satisfaction of the constraint forming the corner.

Then, a little lemma is that cx is decreasing during the algorithm (not necessarily strictly depending on implementation of 4 but at least decreasing - and strictly decreasing with first proposed implementation of 4).

Then, the set D can not have twice the same value in step 8 or 14. Indeed, if algorithm is two time in step 8 in point x_1 and x_2 with a common value for D, it means that w should not have been 0 but at least $x_2 - x_1$. (And $x_2 = x_1$ is impossible because in step 8.d, the algorithm make sure to restart from a point x with $cx < cx_1$).

In the same idea, if the algorithm reaches two time the step 14 in point x_1 and x_2 with a common value for D, then there is a contradiction because $cx_2 < cz < cx_1$. Precisely, it could possible if there is at least two points p, q with $A_D p = A_D q$ and $cp < cq$. But, I take care of removing this possibility by construction of the derived problem.

This is the reason why the derived problem has to be

$$\min_{A_{double} x_{double} + z_{double} g \geq b_{double}, z_{double} \geq 0} z_{double}$$

with g the vector with $g_i = i$ and NOT

$$\min_{A_{double} x_{double} + z_{double} \mathbf{1} \geq b_{double}, z_{double} \geq 0} z_{double}$$

.

So, it can not have more than 2^M step 8 and 14.

Finally, the other step 5 and 8 strictly increases D which is bounded by $\{1, ..., M\}$. So, the algorithm can not perform more than M step 5 or 8 without performing a step 8 or 14. So the algorithm terminates in at most $M2^M$ steps.

4.2 Complexity

The central question related to the complexity of this algorithm is the number of step 8 and 14.

For the step 14, the situation is the following. There is a linear cone $\{x, Ax \geq 0\}$ and a ball centred on c with a radius d tangent to all plane of the cone. If one cut the cone with a plane $\alpha x = 0$, does every plane $A_m x = 0$ from the cone have one (non 0) point in each half space $\alpha x \geq 0$ and $\alpha x \leq 0$?

In the context of the algorithm, it means that as the algorithm will restart from a point x lower than z, is it possible that all plane from D will be seen again (in other D set) ? In low dimension, this is not the case because there is no way to get close to all plane from D without crossing the plane $\{y, cy = cz\}$.

For the step 8, the situation is even harder, because step 8 may correspond to simultaneous multiples corners of step 14. Obviously, it is possible to have a step 8 with $D = \{m_1, ..., m_k\}$, and then, during the following of the process k points with D_1, ..., D_k with $m_i \in D_i$. Yet, the idea of the step 8 is not just to move a little along $-c$ (which will still prevent to see this set again in step 8) but to move a little along $-c$ will keeping a subset of D, and precisely, the subset formed greedily by adding incrementally the rows with larger product with c. This way, the hope is to limit oscillation of plane in D: if a plane is rejected, it can never come back without rejection of one with larger product with c. In other words, it is not possible to have the successively the sets *abcd acd abd* because b should have been keept from *abcd*. This strongly reduces the possible combination seen in step 8. Worse case seems to be the discovery of all planes by increasing order of product with c.

This two points are not most discussed here. Yet, these questions may be interesting because all the presented step are strongly polynomial i.e. complexity (for elementary operation in in $\mathbb{Q}$) is a polynomial in M and N (especially thank to Chubanov algorithm). Thus, if the number of step 8 and 14 was bounded by a polynomial in M. Then the resulting algorithm could be strongly polynomial (in $\mathbb{Q}$).

One interesting point is to see that linear feasibility $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x \geq \mathbf{1}$ (which is an interesting topic in machine learning) is not this far from general linear program $\exists ?x \in \mathbb{Q}^N / \mathcal{A}x \geq b$

References

[1] 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.

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

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

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

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