L'algorithme de la direction améliorante pour la résolution de programmes linéaires

Adrien Chan-Hon-Tong

L'algorithme de la direction améliorante pour la résolution de programmes linéaires

Adrien CHAN-HON-TONG
CEA,LIST,DIASI,LVIC
adrienchanhonton@gmail.com

26 décembre 2013

1 Introduction

La résolution des programmes linéaires c'est à dire des problèmes de la forme

$$\min_{x \in \mathbf{Q}^N} (cx)$$
$$\forall m \in \{1, ..., M\}, a_m x \geq b_m$$

avec a_m, et c des vecteurs $\mathbf{Q}^N$ et b_m des scalaires, est un problème très étudié dans la littérature. Ce problème est un problème polynomial [1]. Mais, les meilleurs bornes polynomiales connues dépendent de la taille L nécessaire pour écrire les données et non pas seulement de $T = \max(N, M)$. Dit autrement, il n'existe aujourd'hui aucun algorithme déterministe connu ayant seulement besoin pour produire la solution d'un nombre d'opérations dans $\mathbb{Q}$ polynomial en M et N.

Il existe cependant des méthodes dont les complexités temporelles ne dépendent que du nombre de contraintes M et du nombre de variables N. Par exemple, la méthode du simplexe [2], très efficace en pratique, qui explore l'ensemble fini des sommets de l'espace des solutions, n'est pas sensible à L. Mais cette méthode n'est pas polynomiale [3] notamment parce que le nombre de sommets augmente exponentiellement avec la dimension de l'espace.

La question de l'existence d'un algorithme possédant ces deux avantages (caractère polynomial et indépendance vis à vis de L) est une question ouverte à ce jour, et, c'est dans ce contexte que s'inscrit cet article. Nous proposons un algorithme dont on conjecture qu'il vérifie les propriétés souhaitées.

2 État de l'art

De nombreuses approches ont été explorées pour tenter de résoudre de façon fortement polynomiale la programmation linéaire. Parmi ces approches,

on trouve :

- la recherche d'une règle de pivotage efficace pour le simplex [4]
- la recherche de solution pour des programmes linéaires ayant certaines spécificités comme l'existence d'une solution binaire [5] ou un nombre de variables par contraintes inférieur à 3 [6]
- la recherche de solution dans le cas où le nombre de caractères nécessaire pour écrire les a est borné [7]
- la recherche de critères d'améliorations pour les approches de type points intérieurs [8] dont on pourra trouver une forme de synthèse dans [9]. Parmi ces approches, certaine propose de chercher la solution pour des formes spécifiques mais néanmoins *complètes* des programmes linéaires [10]. C'est dans ce contexte que cet article se positionne.

Les approches de type points intérieurs se schématise ainsi : à partir d'un point admissible, on cherche à se déplacer étape par étape dans l'ensemble des points admissibles vers un point optimal. Ce type d'approche suppose d'être capable de trouver à chaque étape un déplacement élémentaire admissible et de pouvoir montrer que le nombre d'étapes est borné soit par une borne polynomiale [1] voire par une borne fortement polynomiale. La contribution de cet article est

- de proposer de se déplacer selon deux types de mouvements : des mouvements dans l'orthocentre des contraintes les plus proches du point courant et des sauts selon c
- d'appliquer ces déplacements à une formulation *double primal-dual*

3 Algorithme de la direction améliorante

3.1 Calcul dans $\mathbb{Q}$

Par soucis de simplification, l'algorithme présenté utilise naïvement des opérations dans $\mathbb{R}$ et non dans $\mathbb{Q}$. Typiquement, l'entrée de l'algorithme est constitué de vecteurs unitaires. Pour, $\mathbb{R}$, cela ne restreint pas la généralité car de tous vecteurs non nul, on peut former le vecteur unitaire correspondant par la division par la racine de la norme. Mais ce vecteur unitaire peut ne pas être dans $\mathbb{Q}$.

Néanmoins, cet algorithme semble pouvoir être modifié de sorte que l'ensemble des tests n'utilisent que des calculs dans $\mathbb{Q}$ notamment en comparant les distances au carrées plutôt que les distances. Typiquement, bien que naïvement le calcul de la distance entre un point et un plan fasse intervenir une racine carré, le calcul de la projection du point sur le plan ne fait par contre intervenir que des opérations internes à $\mathbb{Q}$.

3.2 Une forme double primal-dual

Soit

$$\min_{x \in \mathbb{R}^N} (cx)$$
$$\forall m \in \{1, ..., M\}, a_m x \geq b_m$$

un problème primal et soit

$$\max_{y \in \mathbb{R}^M} (c'y)$$
$$\forall m \in \{1, ..., M\}, a'_m y \geq b'_m$$

le dual correspondant. Alors, à la fois le primal et le dual sont équivalents au système d'inéquations suivant sur x et y :

$$\forall m \in \{1, ..., M\}, a_m x \geq b_m$$
$$\forall m \in \{1, ..., M'\}, a'_m y \geq b'_m$$
$$cx = c'y$$

Ce dernier problème est équivalent au primal-dual

$$\min_{x \in \mathbb{R}^N, y \in \mathbb{R}^M, z \in \mathbb{R}} (z)$$
$$\forall m \in \{1, ..., M\}, a_m x + z \geq b_m$$
$$\forall m \in \{1, ..., M'\}, a'_m y + z \geq b'_m$$
$$cx + z \geq c'y$$
$$c'y + z \geq cx$$
$$z \geq 0$$

Ce problème possède un point trivial $x = y = 0, z \geq \max\{b_m, b'_m\}$, et possède un minimum. Sa valeur minimale est 0 si et seulement si le problème d'origine est non vide et possède un minimum.

Cependant, le primal dual du primal dual possède par contre les 3 propriétés : existence d'un point trivial, existence d'un minimum, le minimum est 0.

3.3 Notations

Afin de décrire simplement l'algorithme proposé, introduisons les notations suivantes :

Soit $D \in \mathbb{N}$ avec $D \geq 1$.
Soit $U_D = \left\{x \in \mathbb{R}^D | xx = 1\right\}$
Soit $c \in U_D$
Soit I un ensemble fini avec $|I| \geq 1$ où $|.|$ désigne le cardinal.
Soit a une fonction de I vers U_D et b une fonction de I vers $\mathbb{R}$.
Soit $K = \left\{x \in \mathbb{R}^D | \forall i \in I,\ a(i)\, x \geq b(i)\right\}$

On peut supposer (sec. 3.1) que $\overset{\circ}{K} \neq \emptyset$ et $\forall x \in K$, $cx \geq 0$ et
$\exists x \in K | cx = 0$
Soit $K^* = \left\{ x \in \mathbb{R}^D | \forall i \in I,\ a(i)\, x \geq b(i), cx = 0 \right\}$
L'objectif est de déterminer un point de K^*, il est possible de construire un point trivial de K.

$$\forall x \in K,\ e(x) = \min_{i \in I} (a(i)\, x - b(i))$$

$$\forall x \in K,\ S(x) = \{ i \in I | a(i)\, x - b(i) = e(x) \}$$

$$\forall H \subset I,\ O(H) = \left\{ x \in \mathbb{R}^D | \exists \alpha \in \mathbb{R},\ \forall i \in H,\ a(i)\, x - b(i) = \alpha \right\}$$

$$\forall H \subset I,\ O_0(H) = \left\{ x \in \mathbb{R}^D | \forall i \in H,\ a(i)\, x - b(i) = 0 \right\}$$

$$\forall H \subset I,\ Ker(H) = \left\{ v \in \mathbb{R}^D | \forall i \in H,\ a(i)\, v = 0 \right\}$$

$\forall x \in K$, dans le cas où il existe

$$\lambda(x, v) = \max \left(\{ \mu \in \mathbb{R} | x + \mu v \in K \wedge \forall \nu \in [0, \mu[\ ,\ S(x + \nu v) = S(x) \} \right)$$

Ces notations sont très similaires aux notations standards mais sont utiles pour définir S, O, λ. On prendra garde que désormais, on utilisera des parenthèses et non plus des indices puisque a et b sont maintenant non plus des familles de vecteurs mais des fonctions de domaine fini dans l'ensemble des vecteurs.

3.4 Sortie d'orthocentre

L'algorithme proposé suppose qu'on dispose d'une routine permettant étant donné un ensemble d'hyperplan H et un point x tel que $S(x) = H$, de trouver un point x' tel que cx' est strictement inférieur au minimum des cz pour $z \in O(H)$.

On conjecture qu'un tel point peut être construit simplement soit en ajoutant un vecteur colinéaire à c i.e. $x' = x + \mu c$ (μ étant typiquement négatif), soit si cela n'est pas possible c'est que $x" = x + \lambda(x, c)$ est un point de $K - \overset{\circ}{K}$ (il sature donc $H' \subset H$ et il faut se déplacer selon un vecteur composé de la projection de c sur $O(H)$ et de la projection de c sur $Ker(H')$.

On notera $sorti(x)$ un tel point x' (qui existe forcément si l'orthocentre ne contient pas de solution).

3.5 Synopsis

L'algorithme proposé part d'un point de l'intérieur de l'ensemble des points admissibles (voir sec. 3.1 pour sa construction). L'algorithme fait effectuer à ce point des déplacements élémentaires. Chacun des déplacements améliore le score courant qui est le maximum des scores des points contenus dans la boule maximale centrée sur x c'est à dire $c(x - e(x)\, c)$. Remarquons que le score courant $c(x - e(x)\, c)$ n'est pas ici le score du point courant xc. Chacun des déplacements laisse le point courant dans l'intérieur

de l'ensemble des points admissibles sauf pour atteindre une solution (ce qui correspond à $cx = 0$ voir 3.1).

Étant donné que l'algorithme travaille dans l'intérieur de l'ensemble des points admissibles aucune contraintes n'est saturée (i.e. $a(i)\,x > b(i)$) avant d'atteindre une solution. Ainsi, la notion de contraintes saturées n'est pas pertinente ici mais est remplacée par la notion de contraintes *qui sont les plus proches du point courant*. C'est à dire les contraintes de $S(x)$. Ces contraintes sont appelées les contraintes actives.

Deux types de déplacements sont considérés :

- des déplacements *surs* : des déplacements qui font croître strictement l'ensemble des contraintes actives (au sens de l'inclusion)
- des **sauts** : des déplacements selon c ou le remplacement de x par $sorti(x)$

3.6 Algorithme

algorithme : $x \in \overset{\circ}{K}$

1. si $x \in K^*$ renvoyer x
2. si $x - e(x)\,c \in K^*$ renvoyer $x - e(x)\,c$
3. $\varpi \leftarrow \underset{v \in O(S(x))}{\arg\max} ((c+v)(c+v))$
4. $\omega \leftarrow \underset{v \in Ker(S(x))}{\arg\max} ((c+v)(c+v))$
5. soit s un des éléments de $S(x)$ (exécution invariante à l'élément choisi)
6. $\nu \leftarrow \underset{v \in O(S(x))}{\arg\max} ((a(s)-v)(a(s)-v))$
7. si $\varpi \neq 0$ et $x + \lambda(x,\varpi)\,\varpi \in K^*$, retourner $x + \lambda(x,\varpi)\,\varpi$
8. si $\omega \neq 0$
   (a) retourner $x \leftarrow x + \lambda(x,\omega)\,\omega$
   (b) goto 1
9. si $a(s)\,\nu > c\nu$
   (a) retourner $x \leftarrow x + \lambda(x,\nu)\,\nu$
   (b) goto 1
10. si $\varpi \neq 0$ et $x + \lambda(x,\varpi)\,\varpi \in \overset{\circ}{K}$ et qu'un déplacement selon ϖ augmente le score i.e. $\exists \epsilon > 0$ tel que $xc - e(x) > xc + \epsilon\varpi c - e(x + \epsilon\varpi)$
    (a) retourner $x \leftarrow x + \lambda(x,\varpi)\,\varpi$
    (b) goto 1
11. si $\varpi = 0$ (typiquement $O(H) = \{x\}$)
    (a) $x \leftarrow x - e(x)\,c$
    (b) goto 1

12. $x \leftarrow sortie\,(x)$
13. goto 1

Remarquons que l'utilisation de λ ici justifié en 7, 8.a, 9.a, 10 et 10.a : en 7, 8.a, 10 et 10.a, le déplacement a lieu selon un vecteur ayant un produit scalaire strictement positif avec $-c$ car sinon 0 aurait était meilleur dans 3 et 4. Or si un déplacement selon un vecteur ayant un produit scalaire strictement positif avec $-c$ pouvait être infini, cela signifierait que le score du point courant pourrait être rendu plus petit que 0, or on se place dans le cas où cela est impossible. Pour 9.a, l'argument est similaire mais avec le score de $x - e\,(x)\,c$ et non de x, puisqu'un déplacement selon $\tau\nu$ $(\tau \geq 0)$ augmente le score du point courant mais augmente $e\,(x)$ et ainsi diminue le *score réel* : si $S\,(x)$ est invariant, la différence entre le score courant $x.c - e\,(x)$ et le score après déplacement $xc + \tau\nu c - e\,(x) - \tau a\,(s)\,\nu$ est $\tau\,(c\nu - a\,(s)\,\nu)$ c'est à dire que le score diminue de façon proportionnelle à τ si on se déplace selon ν ce qui permet d'amener le score sous 0, ce qui est impossible.

Remarquons aussi que 3, 4, 6 se ramène à de algèbre linéaire de base : il ne s'agit que de calcul de projection.

4 Discussion

On supposera dans cette discussion que la construction de $sortie\,(x)$ est fortement polynomiale.

4.1 Invariants

Il est clair que toutes les opérations 8.a, 9.a et 10.a, laisse le point courant dans l'intérieur de l'ensemble des points admissibles et améliore strictement le score :

- 8.a améliore à la fois le score et le score du point courant (i.e. $xc - e\,(x)$ et cx puisque cx augmente et que $e\,(x)$ est constant).
- 9.a diminue le score du point courant mais n'a lieu que si le score augmente
- 10.a n'a lieu que si le score augmente.
- pour 8.a et 9.a $e\,(x)$ augmente ou reste égal, donc ne peut pas devenir 0 et 10.a n'a lieu que si le point résultant reste dans l'intérieur de l'ensemble des points admissibles

Donc la seule opération qui peut briser ces invariant est 11. Mais tout d'abord 11 laisse le point courant dans l'intérieur de l'ensemble des points admissibles car sinon c'est qu'il y a un plan de la forme $cz \geq xc - e\,(x)$ dans les contraintes qui est actuellement dans les contraintes actives et donc cela signifie que 2 aurait du avoir lieu.

Ensuite par construction, le score augmente puisque on se déplace selon $-c$.

Ainsi l'algorithme vérifie bien que le point courant est interne et que le score croit.

4.2 Terminaison

Il est clair que le nombre de contraintes est une borne du nombre d'itérations de l'algorithme entre deux sauts (un saut correspondant à l'opération 11) car chaque branchement 8, 9 et 10 augmente strictement le nombre de contraintes dans $S(x)$. Ainsi, la seule chose qui puisse faire que l'algorithme ne termine pas est que 11 ou 12 arrive une infinité de fois.

Mais il n'y a qu'un nombre fini de possibilité pour $S(x)$ (correspondant au nombre de sous ensembles de l'ensemble des contraintes (sans l'ensemble vide)). Donc cela implique qu'un même sous-ensemble réapparaît deux fois au niveau de 11 ou 12. Or on conjecture que soit 8, 9 et 10 aurait alors du s'appliquer lors de la première apparition de cet ensemble qui n'aurait donc pas du atteindre 11 ou 12.

Or il faut se rappeler qu'un déplacement selon c fait **nécessairement** sortir de l'orthocentre courant puisque par construction du problème toutes les contraintes ont un produit scalaire différent avec c.

Donc la terminaison est assuré.

4.3 Nombre d'itérations

On conjecture qu'à chaque saut (étapes 11 ou 12) une contrainte ne réapparaîtra plus.

5 Conclusion

Cet article s'intéresse à la programmation linéaire. On conjecture que l'algorithme présenté résolve la programme linéaire en effectuant un nombre d'opérations dans $\mathbb{Q}$ polynomial en le nombre de contraintes et le nombre de variables. Cet algorithme représente potentiellement une avancée majeure par rapport aux algorithmes polynomiaux actuelles ayant besoin d'un nombre d'opérations dans $\mathbb{Q}$ polynomial en le nombre de contraintes et le nombre de variables et linéaire en la taille nécessaire pour écrire le programme linéaire.

Références

[1] N. Karmarkar, A new polynomial-time algorithm for linear programming, 1984

[2] G. B. Dantzig, Maximization of linear function of variables subject to linear inequalities, 1951

[3] V. Klee et G. J. Minty, How good is the simplex algorithm ?, 1972

[4] J.A. Kelner et D.A. Spielman, A randomized polynomial-time simplex algorithm for linear programming, 2006

[5] S. Chubanov, A strongly polynomial algorithm for linear systems having a binary solution,2012

[6] N. Megiddo, Toward a genuinely polynomial algorithm for linear programming, 1981

[7] E. Tardos, A strongly polynomial algorithm to solve combinatorial linear programs, 1986

[8] M. Bàràsz et S. Vempala, A new approach to strongly polynomial linear programming,2010

[9] I. Lavallée et B.M. Ndiaye et D. Seck,Une approche spécifiquement informatique pour la programmation linéaire,2012

[10] J. Plesník, Deepest point of a polyhedron and linear programming, 2013