Points intérieurs
Introduction

Fabian Bastin
DIRO
Université de Montréal

Vitesse de convergence du simplexe

On sait que la procédure du simplexe converge vers une solution optimale (en supposant qu'au moins une telle solution existe) en un nombre fini d'étapes. Il n'est reste pas moins que ce nombre d'étapes peut être grand.

Peut-il arriver que le simplexe examine toutes les bases réalisables possibles ? La réponse est malheureusement oui.

En général, le simplexe est une méthode rapide. Mais pour une instance donnée, nous n'avons aucune garantie.

Eléments de la théorie de complexité

Complexité : quantité de ressources requises par un calcul.

But : associer à un algorithme des mesures intrinsèques de ses exigences en temps de calcul. Grosso-modo, pour ce faire, nous avons besoin de définir

- une notion de la taille des entrées ;
- un ensemble d'opérations de base ;
- un coût pour chaque opération de base.

Si x est une entrée donnée, le coût de calcul $C(x)$ avec l'entrée x est la somme des coûts de toutes les opérations de base utilisées au cours de ce calcul.

Eléments de la théorie de complexité

Soit $\mathcal{A}$ un algorithme et $\mathcal{J}_n$ l'ensemble de toutes les entrées de taille n. La fonction de coût de pire cas de $\mathcal{A}$ est définie par

$$T^w_{\mathcal{A}}(n) = \sup_{x \in \mathcal{J}_n} C(x).$$

S'il existe une structure de probabilités définie sur $\mathcal{J}_n$, il est possible de définir le coût moyen comme

$$T^a_{\mathcal{A}}(n) = E_{x \in \mathcal{J}_n}[C(x)].$$

où $E_{x \in \mathcal{J}_n}$ est l'espérance sur $\mathcal{J}_n$.

Ce coût moyen est souvent plus difficile à obtenir.

Eléments de la théorie de complexité

Comment sélectionner les trois types d'objet définis plus haut pour l'analyse ?

Pour les algorithmes que nous considérons ici, le choix évident est l'ensemble des quatre opérations algorithmiques de base :
$+, -, \times, /$.

Sélectionner une notion de taille d'entrée et de coût pour les opérations de base dépend du type de données traitées par l'algorithme. Certains types peuvent être représentés à l'intérieur d'une quantité fixée de mémoire informatique, tandis que d'autres nécessitent une mémoire variable.

Eléments de la théorie de complexité

Un concept de base est celui de temps polynomial.

Un algorithme $\mathcal{A}$ est dit algorithme en temps polynomial si $T^w_{\mathcal{A}}(n)$ est bornée supérieurement par un polynôme.

Un problème peut être résolu en temps polynomial s'il existe un algorithme en temps polynomial résolvant le problème.

La notion de temps moyen polynomial est définie similairement, en remplaçant $T^w_{\mathcal{A}}(n)$ par $T^a_{\mathcal{A}}(n)$.

La notion de temps polynomial est généralement prise comme la formalisation de l'efficacité en théorie de la complexité.

La méthode du simplexe n'est pas en temps polynomial

Soit $A \in \mathcal{R}^{m \times n}$, $b \in \mathcal{R}^m$, $c \in \mathcal{R}^n$.

Le nombre d'étapes de pivots est typiquement un petit multiple de n. Toutefois, le nombre d'itérations requises peut être exponentiel.

Une forme de l'exemple de Klee-Minty est

$$\begin{aligned} \max_x \ & \sum_{j=1}^{n} 10^{n-j} x_j \\ \text{t.q. } & 2 \sum_{j=1}^{i-1} 10^{i-j} x_j + x_i \leq 100^{i-1}, \ \ i = 1, \ldots, n \\ & x_j \geq 0, \ j = 1, \ldots, n. \end{aligned}$$

Exemple de Klee-Minty

Considérons le cas $n = 3$.

$$\begin{aligned} \max_x \; & 100x_1 + 10x_2 + x_3 \\ & x_1 \leq 1 \\ & 20x_1 + x_2 \leq 100 \\ & 200x_1 + 20x_2 + x_3 \leq 10000 \\ & x_1 \geq 0, \; x_2 \geq 0, \; x_3 \geq 0. \end{aligned}$$

Sous forme standard, cela donne $m = 3$, $n = 6$, comme nous devons ajouter 3 variables d'écart. On peut montrer que 7 pivots sont nécessaires.

Exemple de Klee-Minty

Sous la forme générale, cela donne $2^n - 1$ pivots.

Pour $n = 50$, cela donne $2^{50} - 1 \approx 10^{15}$. Si on était capable de réaliser un million de pivots par seconde, il faudrait aux environs de 33 ans pour résoudre le problème.

Un peu d'histoire

Adapté de Javier Peña, https://www.stat.cmu.edu/~ryantibs/convexopt-S15/lectures/16-primal-dual.pdf

- Dantzig (1940s) : la méthode du simplexe.
- Klee et Minty (1960s).
- Khachiyan (1979) : premier algorithme en temps polynomial pour la programmation linéaire.
- Karmarkar (1984) : premier algorithme en temps polynomial pour la programmation linéaire.
- Renegar (1988) : algorithme de points intérieurs pour la programmation linéaire basé sur Newton. Meilleure complexité théorique connue à ce jour.

Référence principale : Robert J. Vanderbei, *Linear Programming : Foundations and Extensions*, 5e édition, Springer, 2020, https://link.springer.com/book/10.1007/978-3-030-39415-8

Méthodes de points intérieurs : complémentarité

Pourquoi les problèmes linéaires sont-ils difficiles ? En raison de la complémentarité !

Reprenons la paire primale-duale

$$\begin{aligned} \min_x \ & c^T x \\ \text{s.c. } & Ax = b \\ & x \geq 0, \end{aligned}$$

$$\begin{aligned} \max_\lambda \ & b^T \lambda \\ \text{s.c. } & A^T \lambda + t = c. \\ & t \geq 0 \end{aligned}$$

Nous avions obtenu

$$t_i x_i = 0, \ \ i = 1, \ldots, n.$$

Méthodes de points intérieurs : complémentarité

On peut aussi le voir sur la forme symétrique :

$$\begin{aligned} \min_x \ & c^T x \\ \text{s.c. } & Ax - u = b \\ & x \geq 0, \ u \geq 0. \end{aligned}$$

$$\begin{aligned} \max_\lambda \ & b^T \lambda \\ \text{s.c. } & A^T \lambda + t = c. \\ & \lambda \geq 0, t \geq 0. \end{aligned}$$

Dans ce cas, nous avons

$$t_i x_i = 0, \ i = 1, \ldots, n \quad \lambda_j u_j = 0, \ j = 1, \ldots, m.$$

Notation matricielle

On ne peut écrire $tx = 0$, comme le produit tx est indéfini.

Réécriture :

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} \Rightarrow \begin{pmatrix} x_1 & & & & \\ & x_2 & & & \\ & & x_3 & & \\ & & & \ddots & \\ & & & & x_n \end{pmatrix}$$

Les conditions de complémentarité peuvent alors être réécrites comme

$$XTe = 0, \quad U\Lambda e = 0.$$

Conditions d'optimalité

$$
\begin{aligned}
Ax - u &= b \\
A^T\lambda + t &= c \\
XTe &= 0 \\
U\Lambda e &= 0 \\
x,\ \lambda,\ t,\ u &\geq 0.
\end{aligned}
$$

Ignorons (temporairement) les contraintes de non-négativité. Nous avons $2n + 2m$ équations, à $2n + 2m$ inconnues.

Soucis : le système n'est pas linéaire. La non-linéarité des conditions de complémentarité rend le problème de programmation linéaire fondamentalement plus difficile que la résolution d'un système d'équations linéaires.

Conditions d'optimalité : μ-complémentarité

En plus d'ignorer les contraintes de non-négativité, on va modifier les contraintes de complémentarité en utilisant un paramètre $\mu > 0$:

$$\begin{aligned} Ax - u &= b \\ A^T\lambda + t &= c \\ XTe &= \mu e \\ U\Lambda e &= \mu e. \end{aligned}$$

Les paires de variables primales duales contiennent deux variables de même nature et fixer la valeur d'une variable fixe la seconde (alors qu'une valeur nulle laisse l'autre variable indéterminée).

Direction de recherche

Débuter avec une solution initiale positive (x, λ, u, t).

On va introduire des directions de recherche

$$\Delta x,\ \Delta\lambda,\ \Delta u,\ \Delta t,$$

et on réécrit les équations précédentes avec

$$x + \Delta x,\ \lambda + \Delta\lambda,\ u + \Delta u,\ t + \Delta t,$$

Cela donne

$$\begin{aligned} A(x + \Delta x) - (u + \Delta u) &= b \\ A^T(\lambda + \Delta\lambda) + (t + \Delta t) &= c \\ (X + \Delta X)(T + \Delta T)e &= \mu e \\ (U + \Delta U)(\Lambda + \Delta\Lambda)e &= \mu e. \end{aligned}$$

Direction de recherche

On réarrange avec les variables "Δ" à gauche, les autres termes à droite, et on "jette" les termes non-linéaires :

$$
\begin{aligned}
A\Delta x - \Delta u &= b - Ax + u \\
A^T \Delta\lambda + \Delta t &= c - A^T\lambda - t \\
T\Delta X + X\Delta T &= \mu e - TXe \\
U\Delta\Lambda + \Lambda\Delta U &= \mu e - U\Lambda e.
\end{aligned}
$$

C'est un système linéaire de $2m + 2n$ équations à $2m + 2n$ inconnues, que nous pouvons résoudre, pour définir

$$
\begin{aligned}
x &\leftarrow x + \alpha\Delta x \\
\lambda &\leftarrow \lambda + \alpha\Delta\lambda \\
u &\leftarrow u + \alpha\Delta u \\
t &\leftarrow t + \alpha\Delta t
\end{aligned}
$$

Forcer la convergence

Prendre une plus petite value de μ pour la prochaine itération.

Répéter à partir du début, jusqu'à ce que la solution courante satisfasse, avec une certaine tolérance, les conditions d'optimalité suivantes ;

- Réalisabilité primale : $b - Ax + u = 0$;
- Réalisabilité duale : $c - A^T\lambda - t = 0$;
- Écart de dualité : $b^T\lambda - c^T x = 0$.

Forcer la convergence

Théorème

- *La non-réalisabilité primale devient plus petite d'un facteur* $1 - \alpha$ *à chaque itération.*
- *La non-réalisabilité duale devient plus petite d'un facteur* $1 - \alpha$ *à chaque itération.*
- *Si le primal et le dual sont réalisable, l'écart de dualité diminue d'un facteur* $1 - \alpha$ *à chaque itération (si* $\mu = 0$*, et une convergence légèrement plus lente si* $\mu > 0$*).*

Pas si simple !

L'algorithme travaille itérativement, en calcul un pas à chaque itération, mais comment est-mis à jour le paramètre α ? Comment choisir et mettre à jour μ ?