Multiplication rapide de matrices et applications

Sujet proposé par Frédéric Magniez

magniez[at]lri.fr *Difficulté : facile (*).*

1 Préambule : le contexte

Le premier but de ce projet est d'implémenter astucieusement des algorithmes rapides de multiplication de matrices. Le premier est dû à Straussen et son amélioration à Winograd. Si la multiplication naïve de matrices carrées de taille N requiert $O(N^3)$ *opérations* (*i.e.* additions et multiplications), il est possible d'utiliser seulement $O(N^{2.81})$ opérations par une technique de division pour régner. En poursuivant dans cette direction, il serait possible de considérer des algorithmes plus sophistiqués n'utilisant que $O(N^{2.38})$ opérations, mais cela est en dehors de ce projet.

Ensuite, nous aborderons des problèmes connexes utilisant en sous-routine la multiplication de matrices et parfois des techniques aléatoires. Le premier consiste à vérifier qu'un produit de matrice est correct avec seulement $O(N^2)$ opérations. Il s'agit du test probabiliste de Freivalds.

Le deuxième est le calcul de l'inverse d'une matrice à l'aide de nos algorithmes de multiplication de matrices. En en déduira un algorithme pour tester si un déterminant est nul. Une application est la détection d'un couplage parfait dans un graphe biparti. Cet application est étroitement reliée au théorème de Schwartz sur le nombre de zéros d'un polynômes à plusieurs variables dans un corps fini. Cet algorithme sera comparé à celui de Ford-Fulkerson qui trouve un couplage maximal en utilisant une technique de maximisation de flot.

Enfin, nous résoudrons le problème de détection de k-cycles dans un graphe non orienté toujours à l'aide de nos algorithmes de multiplication de matrices.

2 Détail du sujet

2.1 Interface

L'interface n'est pas un objectif dans ce projet. Toutefois, pour la saisie des matrices ou du graphe (biparti ou non), le choix sera laissé à l'utilisateur entre un tirage aléatoire, la lecture d'un fichier, ou une saisie manuelle.

Dans le cas d'un fichier, la première ligne du fichier contiendra la taille des matrices, puis l'énumération des lignes de chaque matrice. Pour les graphes, en première ligne sera le nombre de nœuds considérés, puis sur chaque ligne du fichier la liste des successeurs de chaque nœud.

2.2 Algorithmes

On supposera toujours que les matrices sont de taille $N = 2^n$. Dans votre projet vous pourrez considérer ou expliquer comment considérer le cas général.

A part pour le calcul de l'inverse d'une matrice, on pourra toujours se placer sur les entiers afin d'accélérer le temps de calcul et réduire l'espace mémoire.

2.2.1 Multiplication de matrices

Les algorithmes de multiplication rapide sont basés sur une observation dans le cas des matrices 2×2, qui se généralise aisément par induction lorsqu'on découpe une matrice M de taille N en 4 sous-blocs de taille $N/2$:

$$M = \begin{pmatrix} M11 & M12 \\ M21 & M22 \end{pmatrix}.$$

Voici maintenant les décompositions récursivent qu'utilisent respectivement les algorithmes de Strassen [Str69] et Winograd [Win71]. On ne manquera pas de les justifier.

La matrice C est le produit de deux matrices A et B. Les autres matrices sont des matrices intermédiaires utilisées dans le calcul.

Pour Strassen :

$$\begin{aligned} M1 &= (A11 - A22) * (B21 + B22) & M2 &= (A11 + A22) * (B11 + B22) \\ M3 &= (A11 - A21) * (B11 + B12) & M4 &= (A11 + A12) * B22 \\ M5 &= A11 * (B12 - B22) & M6 &= A22 * (B21 - B11) \\ M7 &= (A21 + A22) * B11 & & \end{aligned}$$

$$\begin{aligned} C11 &= M1 + M2 - M4 + M6 & C12 &= M4 + M5 \\ C21 &= M6 + M7 & C22 &= M2 - M3 + M5 - M7 \end{aligned}$$

Pour Winograd :

$$\begin{aligned} S1 &= A21 + A22 & S2 &= S1 - A11 \\ S3 &= A11 - A21 & S4 &= A12 - S2 \\ S5 &= B12 - B11 & S6 &= B22 - S5 \\ S7 &= B22 - B12 & S8 &= S6 - B21 \end{aligned}$$

$$\begin{aligned} M1 &= S2 * S6 & M2 &= A11 * B11 \\ M3 &= A12 * B21 & M4 &= S3 * S7 \\ M5 &= S1 * S5 & M6 &= S4 * B22 \\ M7 &= A22 * S8 & & \end{aligned}$$

$$T1 = M1 + M2 \qquad T2 = T1 + M4$$

$$\begin{aligned} C11 &= M2 + M3 & C12 &= T1 + M5 + M6 \\ C21 &= T2 - M7 & C22 &= T2 + M5 \end{aligned}$$

Dans un premier temps, la récursion s'arrêtera lorsque les matrices sont de taille 1. On comparera ces algorithmes avec l'algorithme standard. La vérification proprement dite du calcul s'effectuera avec la stratégie définie dans le paragraphe suivant. Cette comparaison mettra en évidence un seuil pour chacun des algorithmes sur la taille des matrices à partir duquel la multiplication standard est moins rapide. En dessous, de ce seuil on arêtera la récursion et on calculera de manière standard le produit.

2.2.2 Vérification

Freivalds [Fre79] a montré que si deux matrices A et B étaient différentes, alors $AX \neq BX$ avec probabilité au moins $1/2$ lorsque X est un vecteur aléatoire dont chaque coordonnée est soit 0, soit 1 de manière équiprobable.

Pour vérifier que la matrice $C = A \times B$, il suffit de vérifier pour un tel X que $CX = A(BX)$, pour suffisamment de vecteurs X. Vous justifierez que le nombre d'opérations utilisées pour chaque vérification est en $O(N^2)$.

Si $C = A \times B$, alors on a toujours $CX = A(BX)$, si $C = A \times B$, et qu'on répète T fois cette vérification avec T vecteurs tirés au sort, alors la probabilité d'avoir $CX \neq A(BX)$ au moins une fois est au moins $1 - 1/2^T$. La probabilité de se tromper est donc de $1/2^T$.

2.2.3 Inverse

Dans cette partie il conviendra de se placer sur les réels.

Supposons A inversible, alors $A^{-1} = (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}}$. Donc il est suffisant de savoir inverser les matrices symétriques définies positives. Si A est ainsi, alors A^{-1} se déduit de A par induction sur son découpage en blocs ainsi :

$$A = \begin{pmatrix} B & C^{\mathrm{T}} \\ C & D \end{pmatrix} \implies A^{-1} = \begin{pmatrix} B^{-1} + B^{-1}C^{\mathrm{T}}S^{-1}CB^{-1} & -B^{-1}C^{\mathrm{T}}S^{-1} \\ -S^{-1}CB^{-1} & S^{-1} \end{pmatrix},$$

avec $S = D - CB^{-1}C^{\mathrm{T}}$.

Une autre manière de procéder à la Strassen est la suivante, où dans ce qui suit $B = A^{-1}$:

$$\begin{aligned} P &= (A11)^{-1} & Q &= A21 * P \\ R &= P * A12 & S &= A21 * R \\ T &= S - A22 & U &= T^{-1} \end{aligned}$$

$$\begin{aligned} B12 &= R * U & B21 &= U * Q \\ B22 &= -U \end{aligned}$$

$$V = R * B21$$

$$B11 = P - V$$

On testera chacun des algorithmes implémentés en adaptant le test de Freivalds exposé précédemment.

Puis on adaptera ces algorithmes pour détecter si une matrice est inversible, *i.e.* si son déterminant est non nul.

2.2.4 Couplage parfait

Un graphe equibiparti est la donnée d'un triplet $G = (A, B, E)$, où A et B sont des ensembles de sommets disjoints et de tailles égales, et E un ensemble d'arêtes entre A et B. Nous supposerons ici que les arêtes ne sont pas orientées. Un *couplage parfait* C est un sous-ensemble de E qui *marie* chaque sommet de A avec un seul sommet de B. On présente ici l'approche de Mulmuley, Vazirani et Vazirani [MVV87].

On considère des variables x_{ij}, pour $1 \leq i, j \leq N$. Considérons la matrice M de taille $N \times N$ telle que $M_{ij} = x_{ij}$ si l'arête (i, j) est dans E, et $M_{ij} = 0$ sinon.

Le déterminant $\det M$ de M est donc un polynôme en (x_{ij}) de degré N. Ce polynôme est formé des monômes de la forme $x_{1\sigma(1)}x_{2\sigma(2)} \cdots x_{N\sigma(N)}$, où σ est une permutation de $\{1, 2, \ldots, N\}$ quelconque. Plus précisément ce monôme apparaît dans $\det M$ si et seulement si chacune des arêtes $(1, \sigma(1)), (2, \sigma(2)), \ldots, (N, \sigma(N))$ est dans E, qui est alors un couplage parfait du graphe. Donc $\det M$ n'est pas le polynôme nul si et seulement si G admet un couplage parfait.

Le lemme de Schwartz est alors ici utilisé pour vérifier algorithmiquement et aléatoirement que ce polynôme n'est pas nul. Soit P un polynôme non nul de degré d à m variables (sur un corps quelconque).

Ce théorème dit que si à chaque variable est assignée à une valeur prise uniformément au hasard dans un ensemble S de taille s, alors P ne s'annule pas en ces valeurs avec probabilité au moins d/s.

On comparera les résultats obtenus avec l'algorithme usuel pour calculer le couplage parfait d'un graphe biparti de Ford-Fulkerson. Cette méthode utilise la notion de flots pour calculer un couplage de taille maximale. S'il contient tous les nœuds alors il est parfait. Pour une description de l'algorithme, on pourra se reporter à la pâle HC 2003-04 : `http://www.enseignement.polytechnique.fr/informatique/IF/04/hc/hc.html`.

2.2.5 Cycles

Si A est la matrice d'adjacence d'un graphe (non orienté), le calcul de A^k sur l'algèbre booléenne permet de déterminer les paires de sommets (i, j) reliés par un chemin de taille plus petite que k : $(A^k)_{i,j} = 1$ ssi un tel chemin existe entre i et j.

Par contre, si A est la matrice d'adjacence d'un graphe orienté acyclique (*i.e.* sans cycles), alors $(A^k)_{i,j} = 1$ ssi un chemin existe de longueur exactement k de i à j.

Etant donné un paramètre fixe k et un graphe non orienté G, on aimerait déterminer un cycle *simple* (*i.e.* sans sous-cycles) de taille exactement k, et en trouver un s'il en existe, en utilisant l'approche d'Alon, Yuster et Zwick [AYZ95]. Pour ce, on va orienter aléatoirement G de manière à le rendre acyclique. Soit $\vec{G}$ le graphe G orienté. Puis il suffira de vérifier s'il existe une paire (i, j) connectée par un chemin de longueur $(k - 1)$ de i à j dans $\vec{G}$, telle que (i, j) est une arête du graphe initial G non orienté.

L'orientation est effectuée en tirant au sort une permutation σ de V dans $\{1, 2, \ldots, N\}$, où N est la taille de V. Alors, une arête (i, j) de G sera une arête de $\vec{G}$ de i vers j si $\sigma(i) < \sigma(j)$, et de j vers i sinon.

On peut montrer que la probabilité de trouver un cycle, s'il en existe, est de $1/k!$. Il suffit donc de repéter T fois l'expérience, en tirant une nouvelle permutation à chaque itération, pour obtenir une probabilité d'erreur inférieure à $(1 - 1/k!)^T$.

Le calcul de A^k se fera en $O(\log k)$ multiplications de matrices en utilisant l'algorithme d'exponentiation rapide standard. Même s'il n'est pas possible d'adapter les algorithmes de multiplication rapide de Strassen et Winograd, il suffira de faire les multiplications sur les entiers et d'associer chaque coefficient non nul au booléen 1.

2.3 Structure de données

La représentation des matrices est fondamentale dans ce projet. Une représentation de type `array` n'est pas adaptée à la récursivité des méthodes utilisées. Une représentation récursive en forme d'arbre de *rang* 4 (chaque nœud a exactement 4 fils) conviendra mieux. On définira une classe `Matrice` contenant cinq variables :

- `int n` : la profondeur dans l'arbre depuis les feuilles, *i.e.* la taille de la matrice est 2^n,
- `Matrice M11, M12, M21, M22` : les 4 sous-matrices correspondants aux blocs haut-gauche, haut-droit, bas-gauche, bas-droit,
- `int val` : la valeur de l'entrée de la matrice (lorsqu'on est sur une feuille de l'arbre, *i.e.* $n = 0$). Pour la partie inversion de matrice, on utilisera un type `float` pour `val`.

On écrira une méthode permettant de passer de la représentation tableau à la représentation récursive, et inversement.

3 Pour aller plus loin

Tout d'abord, une bonne référence autour de thèmes reliés est le livre d'Aho, Hopcroft et Ullman [AHU75].

Avant de considérer l'une de ces extensions, il est fortement conseillé de prendre contact.

Des algorithmes plus performant mais plus sophistiqués pour la multiplication de matrices existent. Ils n'utilisent que $O(N^{2.5})$ opérations [CW81] ou encore $O(N^{2.38})$ opérations [CW87]. Hélas leur implémentation est délicate.

Lorsque les matrices sont creuses, il y a des algorithmes meilleurs qui atteignent de meilleurs bornes [YZ04].

Tous ces algorithmes restent valident sur un corps quelconque. On pourra les adapter pour les corps finis, ou encore sur les rationnels sans avoir à passer par le type des réels. Les décompositions standards (LU, LUP...) ont le même coût que la multiplication rapide. L'algorithme utilise une version récursive par blocs de la décomposition [AHU75]. Il est alors possible de calculer valeurs propres, déterminant, et inverse dans la foulée.

La recherche de couplage parfait peut être généralisée à un graphe quelconque soit en utilisant encore la multiplication de matrices [MVV87], soit en utilisant la technique de maximisation de flots [MV80].

Enfin, pour la détection de k-cycles, il y a une amélioration consistant à colorier aléatoirement le graphe avec k couleurs plutôt qu'en l'orientant. On gagne au niveau de la probabilité de succès puisque $k!$ est alors remplacé par 2^k [AYZ95]. Dans le cas de petits cycles comme le triangle, ou si le cycle est de taille paire ou encore un multiple de 4, il y a d'autre stratégies plus rapide [AYZ97].

L'adresse `http://www.enseignement.polytechnique.fr/profs/informatique/Frederic.Magniez/04-05/INF_431/projet.html` contient certains des articles référencés. Si nécessaire, des indications complémentaires y seront ajoutées au fur et à mesure.

Références

[AHU75] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The design and analysis of computer algorithms*. Reading, Mass. : Addison-Wesley Pub. Co., 1975 printing, 470 p. CALL NUMBER : QA76.6 .A36, 1975.

[AYZ95] Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *Journal of the ACM*, 42(4) :844–856, July 1995.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17(3) :209–223, March 1997.

[CW81] D. Coppersmith and S. Winograd. On the asymptotic complexity of matrix multiplication. *SIAM Journal of Computing*, 11 :472–492, 1981.

[CW87] Coppersmith, D. and Winograd, S. Matrix multiplication via arithmetic progression. In *Proc. 19^{th} Annual ACM Symposium of Theory of Computing*, pages 1–6, 1987.

[Fre79] Rūsiņš Freivalds. Fast probabilistic algorithms. In J. Bečvář, editor, *Mathematical Foundations of Computer Science 1979*, volume 74 of *lncs*, pages 57–69, Olomouc, Czechoslovakia, 3–7 September 1979. Springer-Verlag.

[MV80] S. Micali and V. Vazirani. An algorithm for maximum matching in general graphs. In *Proceedings of the 21st IEEE Annual Symposium on the Foundations of Computer Science*, pages 17–27, 1980.

[MVV87] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7 :105–113, 1987.

[Str69] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13 :354–356, 1969.

[Win71] S. Winograd. On multiplication of 2×2 matrices. *Linear Algebra Appl.*, 4 :381–388, 1971.

[YZ04] Yuster and Zwick. Fast sparse matrix multiplication. In *ESA : Annual European Symposium on Algorithms*, 2004.