CARI'12

Algorithmes matriciels pour les graphes

Algorithmes matriciels pour la construction d'une matrice des cycles fondamentaux d'un graphe

Guesmi Hela [1], Hasni Hamadi [2] & Mahjoub Zaher [3]

[1,3] Université de Tunis El Manar, Faculté des Sciences de Tunis
Campus Universitaire - 2092 Manar II - Tunis, Tunisie
[2] Ecole Nationale des Sciences de l'Informatique, 2100 Manouba, Tunisie

guesmihela@yahoo.fr, hamadi.hasni@ensi.rnu.tn & zaher.mahjoub@fst.rnu.tn

RÉSUMÉ. Nous étudions le problème de la construction d'une matrice des cycles fondamentaux d'un graphe en adoptant une approche se ramenant à la résolution d'un système matriciel triangulaire creux. Partant dune version standard structurée en un nid de boucles, nous dérivons d'autres versions via la technique de permutation de boucles afin d'assurer une meilleure localité des données et un déplacement d'invariant de boucles. Nous dérivons ensuite d'autres versions en utilisant les formats de compression pour matrices creuses MSC et FSA. Nous établissons enfin une étude de performance comparative inter-algorithmes. Une série d'expérimentations effectuées sur divers graphes réels, aléatoires et de bancs d'essai permettent de valider l'étude théorique.

ABSTRACT. In this paper we study the problem of constructing a fundamental cycle matrix in a graph by adopting an approach that leads to solving a sparse triangular matrix system. Starting with a standard loop nest structured version, we derive other versions by applying the loop interchange technique in order to ensure a better data locality and loop invariant motion. We then derive other versions by using the MSC and FSA sparse matrices compressing formats. We furthermore establish an inter-version comparative performance study. Finally, a series of experiments on a set of real, random and benchmark graphs is achieved in order to validate the theoretical study.

MOTS-CLÉS : Matrice des cycles fondamentaux, matrice creuse/triangulaire, nid de boucles, permutation de boucle, stockage compressé.

KEYWORDS : Compressing format, fundamental cycle matrix, loop interchange, loop nest, sparse/ triangular matrix.

1. Introduction

L'analyse de l'espace des cycles d'un graphe a plusieurs applications dans le monde réel par exemple. en génie électrique, analyse structurelle, biologie, chimie, reconstruction de surfaces, etc. L'espace des cycles d'un graphe peut être défini par une base de cycles ou la matrice des cycles associée [12]. On s'intéresse particulièrement ici à la matrice des cycles fondamentaux (MCF) d'un graphe car elle nous sert comme élément important d'une procédure de prétraitement pour la résolution du problème du flot de coût minimum dans un réseau, via une approche orientée flots sur les cycles et non flots sur les arcs comme c'est classiquement le cas [9].

Il se trouve que plusieurs problèmes sur les graphes peuvent être résolus par des algorithmes matriciels. En effet, ces derniers présentent une souplesse au niveau de l'implémentation, comparativement aux algorithmes qui se basent sur l'aspect graphe et utilisent souvent des structures de données assez sophistiquées [8].

La suite du papier est organisée comme suit. Dans la section 2, nous présentons la formulation théorique du problème. La section 3 est dévolue à l'étude de l'algorithme standard structuré en un nid de boucles à partir duquel plusieurs versions sont dérivées. Par la suite, les versions associées à deux formats de compression pour matrices creuses i.e. MSC (Modified Storage Column) et FSA (Format de Stockage Adapté) sont décrites. Dans la section 4, on procède à la présentation d'une étude expérimentale effectuée sur divers graphes et ce, afin d'établir une évaluation de performances comparatives entre les différents algorithmes conçus.

2. Présentation du problème

Considérons un graphe connexe orienté G ayant n nœuds et m arcs numérotés arbitrairement. Soit c=m-n+1 le nombre cyclomatique de G. Une base de cycles fondamentaux (BCF) de G est constituée de c cycles. La matrice des cycles fondamentaux (MCF) associée est une matrice de taille (c,m) à éléments dans {-1,0,1} où chaque ligne représente un cycle de la base [3][4].

Soit $\overline{A}$ la matrice d'incidence nœud-arc du graphe G privée de sa dernière ligne. Partant d'un arbre de recouvrement (AR), la matrice $\overline{A}$, de dimension (n-1,m), est décomposée, via des permutations éventuelles de ses colonnes, comme suit : $\overline{A} = [\overline{A}_1 | \overline{A}_2]$ où $\overline{A}_1$ est une matrice carrée (n-1,n-1) associée à l'AR et $\overline{A}_2$ est une matrice (n-1,c) associée au coarbre correspondant. La MCF, notée C, est décomposée comme suit $C=[C_1|C_2]$ avec C_1 (resp. C_2) est de dimensions (c,n-1) (resp. (c,c)). En choisissant pour chaque cycle fondamental une orientation analogue à celle de la corde

associée, C_2 est dans ce cas la matrice identité. Grâce à la propriété d'orthogonalité cycle-cocycle [3] on obtient: $\overline{A}\, C^T = 0 \Rightarrow \overline{A}_1\, C_1^T + \overline{A}_2 = 0$ (1)

A partir de (1), on déduit $C_1 = - \overline{A}_2 {}_T(\overline{A}_1^{-1})^T$. La détermination de C_1 revient donc à inverser la matrice $\overline{A}_1$, puis à calculer le produit matriciel $\overline{A}_2^T\, (\overline{A}_1^{-1})^T$ [3]. Toutefois, une approche alternative et moins coûteuse consiste à résoudre le système matriciel suivant [9] : $\overline{A}_1\, C_1^T = - \overline{A}_2$ (2)

Il s'agit donc de résoudre c systèmes linéaires de taille n-1 où chaque système permet de déterminer une colonne (resp. ligne) de C_1^T (resp. C_1). Notons que $\overline{A}_1$ (sous matrice de $\overline{A}$ qui est creuse) et a au plus deux éléments non nuls (-1, 1) par colonne. Une renumérotation adéquate de ligne et colonnes de $\overline{A}_1$ conduit à une structure triangulaire [6]. On aura donc affaire à un système matriciel triangulaire creux.

3. Résolution du système matriciel triangulaire creux – Etude théorique

Par mesure de simplification, notons AC=B le système matriciel triangulaire (SMT) à résoudre où A est une matrice triangulaire inférieure de taille (n-1), C et B deux matrices rectangulaires (n-1,c). Dans le but d'optimiser la complexité spatio-temporelle de la résolution, nous allons exploiter le caractère creux de A.

3.1. Format DNS

Divers travaux sur la résolution de système triangulaires creux issus de la factorisation LU sont connus [7]. Toutefois, le problème creux qui nous intéresse est très particulier (voir section 2). C'est ainsi que nous allons utiliser la version standard de l'algorithme de résolution du système, notée KIJ, qui a la structure d'un nid non parfait de trois boucles (voir ci-dessous) [5]. Nous dérivons ensuite de meilleures versions via la technique de permutation de boucles [5][10]. C'est la version KJI qui a été d'abord conçue. Elle tient compte du fait chaque colonne ne contient que deux éléments non nuls, ce qui permet de réduire le nombre de tests sur les éléments de A.

La permutation combinée avec la technique efficace de déplacement d'invariant de boucle [1] a été ensuite appliquée, d'où les versions IJK et JIK. Le test sur l'élément A(i,j) de A qui était au troisième niveau (dans KIJ et KJI) devient un invariant de boucle dans IJK et JIK et peut être remonté au second niveau, d'où une réduction du nombre de tests logiques, soit n^2 au lieu de cn^2. Soulignons que tous ces algorithmes (ainsi que les suivants de la section 3.2.2) requièrent **O(cn)** i.e. **O(mn)** opérations arithmétiques. On explicite ci-dessous à titre illustratif les deux algorithmes KIJ et IJK.

```
Version KIJ
Pour k=1,c
      C(1,k)=B(1,k) /A(1,1)
      Pour i=2, n-1
            Pour j=1,i-1
                Si (A(i,j)≠0) alors
                  B(i,k)=B(i,k)-A(i,j)*C(j,k)
                 Finsi
            Finpour
            C(i,k)=B(i,k)/A(i,i)
      Finpour
Finpour
```

```
Version IJK
Pour i=1,n-1
      Pour j=1,i-1
            Si (A(i,j)≠0) alors
              Pour k=1,c
                    B(i,k)=B(i,k)-A(i,j)*C(j,k)
               Finpour
            Finsi
      Finpour
     Pour k=1,c
           C(i,k)=B(i,k)/A(i,i)
      Finpour
Finpour
```

Sur un autre plan, nous avons utilisé l'optimisation du paramètre dit nombre de défaut de cache et une amélioration de la localité spatiale des données [13]. Ceci requiert que les données soient traitées selon le mode où elles sont stockées.

Nous donnons dans le tableau suivant, pour chacune des matrices A, B et C et chaque version, le nombre d'accès (NA), le nombre de tests logiques (NT) et le mode d'accès (MA) ainsi que le noyau d'algèbre linéaire correspondant [5].

Version	A			B		C		Noyau
	NA	NT	MA	NA	MA	NA	MA	
KIJ	cn^2	cn^2	ligne	cn^2	colonne	cn^2	colonne	DOT
KJI	cn^2	cn^2	colonne	cn^2	colonne	cn^2	colonne	GAXPY
IJK	cn^2	n^2	ligne	cn^2	ligne	cn^2	ligne	GAXPY
JIK	cn^2	n^2	colonne	cn^2	ligne	cn^2	ligne	AXPY

Tableau 1. *Tableau récapitulatif DNS*

A partir du tableau 1, on note dans la version IJK des accès ligne pour les trois matrices et un noyau GAXPY [5]. De plus, le déplacement de l'invariant de boucles a réduit le nombre de tests par un facteur c. Si le mode de stockage *ligne* des matrices est adopté (en langage C), il est possible en se basant ici sur les quatre critères NA, NT, MA et noyau, de classer théoriquement les quatre algorithmes comme suit : IJK, JIK, KIJ, KJI.

3.2. Formats compressés

3.2.1. Présentation des formats de compression

Plusieurs formats de compression (FC) pour matrices creuses sont connus dans la littérature. Citons en particulier les formats généralistes CSR, CSC, COO. Dans le cas de matrices creuses triangulaires inférieures, on relève les formats MSC et MSR [10].

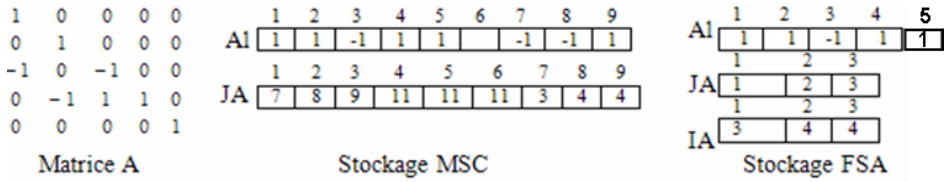
Soit nnz le nombre d'éléments non nuls de la matrice A (de taille n-1) avec nnz <=2n-2. Notons que pour notre problème et pour le stockage MSC, la complexité spatiale est de 2nnz +2 entiers, soit 4n-2 entiers au plus.

Nous proposons dans ce qui suit un format adapté à notre matrice et dénommé FSA (Format de Stockage Adapté). Dans FSA, on stocke uniquement les éléments diagonaux de la matrice dans un tableau Al. On parcourt les éléments de la matrice par colonne et on retient les numéros de colonnes et lignes qui correspondent à un élément non nul (opposé à celui de la diagonale d'une même colonne). Ainsi, on stocke 2nnz-n entiers, soit 3n-4 entiers au plus. Trois tableaux sont en fait utilisés comme suit.

- Al est un tableau d'entiers de taille n-1 où l'on stocke les éléments diagonaux.

- JA est un tableau d'entiers de taille nnz-n qui contient le numéro de colonne de chaque élément non diagonal ; IA est un tableau d'entiers de taille nnz-n qui contient le numéro de ligne de chaque élément non diagonal.

Les formats MSC et FSA sont illustrés ci-après pour une matrice où n-1=5 et nnz =8.



3.2.2. Algorithmes pour les formats MSC et FSA

Ce sont les versions KIJ (version initiale) et IJK (meilleure en théorie) qui ont été retenues pour en dériver les versions *compressées* correspondantes. Soulignons que seule la matrice A est compressée, les matrices B et C ne le sont pas. En effet, dans le but de réduire encore plus la complexité spatiale de nos algorithmes, C est initialisée à B. Les calculs se font donc directement sur C. Comme la structure de C est a priori inconnue, elle est stockée en format dense. De plus, afin d'optimiser nos algorithmes, la technique du remplacement scalaire est appliquée pour réduire le nombre d'accès (directs et indirects) aux éléments. On remplace ici un élément de tableau (accès mémoire indirect) par un scalaire, d'où un accès plus rapide. Cette optimisation est recommandée dans le cas où l'instruction en question se trouve dans d'une boucle [10].
Dans la version IJK_FSA, on a jugé inutile d'utiliser une troisième boucle j puisque le corps 'C(j,k)= C(j,k)+Al(JA(i))*C(JA(i),k)' est exécuté une seule fois. Connaissant ainsi la position de l'élément de C qu'on va modifier, la boucle j est réduite à une seule itération i.e. j=IA(i). Pour le stockage FSA, rappelons qu'on a stocké les éléments diagonaux de A seulement dans le vecteur Al. De ce fait, pour chaque colonne j de A, l'élément non nul A(i,j) est récupéré par l'élément '-Al(j)'. Notons alors que dans l'algorithme IJK_FSA, l'instruction finale est 'C(j,k)=C(j,k)+Al(JA(i))*C(JA(i),k)' qui traduit

'C(j,k)=C(j,k)-(-Al(JA(i)))*C(JA(i),k)'. Dans IJK_FSA, et IJK_MSC nous avons utilisé la technique du remplacement scalaire (avant la boucle k).

```
IJK_ MSC  / Données: Al, JA /
Pour i=1, n-1
      Pour j=JA(i), JA(i+1)-1
           l= JA(j)
           Pour k=1,c
                C(i,k)=C(i,k)/Al(i)
                C(l,k)= C(l,k)-C(i,k)*Al(j)
          Finpour
      Finpour
Finpour
```

```
IJK_FSA / Données : Al, JA, IA /
Pour  i=1, n-1
      j=IA(i), l= JA(i)
      Pour k=1, c
           C(i,k)=C(i,k)/Al(i)
           C(j,k)= C(j,k)+Al(l)*C(l,k)
      Finpour
Finpour
```

Nous donnons dans le tableau suivant, pour C et chaque version, le nombre d'accès (NA), le mode d'accès (MA) ainsi que le noyau d'algèbre linéaire [5]. Pour A, nous donnons le nombre d'accès aux vecteurs Al, JA et IA.

Version	A			C			Noyau
	NA_Al	NA_JA	NA_IA	NA	NA	MA	
KIJ_FSA	cn	cn	cn	cn	cn	col	DOT
KIJ_MSC	cn	cn	--	cn	cn	col	DOT
IJK_MSC	cn	cn	--	cn	cn	ligne	GAXPY
IJK_FSA	cn	cn	cn	cn	cn	ligne	GAXPY

Tableau 2. *Tableau des modes/nombres d'accès*

Notons que pour KIJ_MSC et KIJ_ASF, nous avons un accès ligne à A, un accès colonne à C et un noyau DOT. Pour IJK_MSC et IJK_ASF on accède à A en ligne de même à C et on conserve le noyau GAXPY. En comparant KIJ_MSC et IJK_MSC, nous déduisons que la seconde est meilleure puisqu'on a une meilleure localité des données (en nombre d'accès ligne). Il en est de même pour KIJ_FSA et IJK_FSA. Le classement théorique est donc pour un même stockage IJK puis KIJ.

4. Résultats expérimentaux

Dans le but d'avoir des résultats significatifs, trois types de graphes ont été testés : réels (5), bancs d'essai (2) et générés (6). Nous nous limitons ici aux expérimentations sur les 6 derniers générés par NETGEN [11]. Nos tests ont été effectués sur un PC Acer (Intel Pentium Core 2 Duo, 2.4 GHz, RAM 1 Go) OS Linux (Ubuntu 10.04). Les temps mesurés sont la moyenne de dix passages et sont exprimés en ms.

				Stockage DNS				Stockage compressé			
G	**n**	**m**	**c**	**KIJ**	**KJI**	**IJK**	**JIK**	**KIJ_MSC**	**KIJ_FSA**	**IJK_MSC**	**IJK_FSA**
G1	300	1018	719	118.24	133.25	**3.66**	3.73	9.28	7.85	4.35	**4.15**
G2	500	2030	1531	700.35	863.84	**12.76**	13.42	34.57	29.71	15.65	**14.93**
G3	500	3033	2534	1122.59	1374.94	**20.74**	21.61	58.37	50.18	25.64	**24.05**
G4	500	5028	4529	2028.54	2524.38	**36.67**	38.49	103.16	89.03	45.66	**43.71**
G5	500	10002	9503	4269.34	4939.58	**76.20**	77.05	236.03	278 .9	98.43	**89.96**
G6	1000	10000	9001	10576,5	12361.84	**145.24**	148.60	415.21	367 .78	184.91	**172.88**

Tableau 3. *Temps d'exécution (en ms) des différents algorithmes*

A partir du tableau 3, et en considérant les quatre versions en stockage DNS, la version IJK s'avère être (comme présumé) la plus rapide, suivie, dans l'ordre, par les versions JIK, KIJ et enfin KJI. Cela confirme donc notre étude théorique.
Concernant les versions en stockage MSC et FSA, le meilleur temps est celui de IJK_FSA, suivie, dans l'ordre, par IJK_MSC, KIJ_FSA, KIJ_MSC. Ceci est conforme avec notre étude théorique pour un même mode de stockage.
Notons que pour les versions en DNS, IJK est de 32.3 à 72.75 (resp. 1.03 à 1.05) fois plus rapide que KIJ (resp. JIK).
Pour les versions en formats compressés, IJK_FSA est de 1.04 à 1.07 (resp. 1.87 à 2.13) fois plus rapide que IJK_MSC (resp. KIJ_FSA). De plus, IJK_MSC est de 2.27 à 2.45 fois plus rapide que KIJ_MSC.
Enfin, IJK en DNS est de 1.19 à 1.23 fois plus rapide que IJK_FSA.
Nous pouvons donc dire que si nous négligeons la complexité spatiale, la version IJK en DNS l'emporte. Si nous en tenons compte et retenons IJK_FSA, nous devons accepter un accroissement en temps d'exécution qui va de 13 à 30% dans les cas traités.
Soulignons que l'impact de la technique de déplacement d'invariant de boucle (dans les versions DNS) a été primordial pour classer IJK en première position.

5. Conclusion

Nous nous sommes intéressés dans ce papier à un problème de théorie des graphes ayant diverses applications pratiques, à savoir le calcul d'une matrice des cycles fondamentaux d'un graphe et ce, selon une approche se ramenant à la résolution d'un système matriciel triangulaire creux. A partir d'un algorithme initial, des techniques efficaces d'optimisation d'algorithmes structurés en nids de boucles (permutation de boucle, amélioration de la localité des données, déplacement d'invariant de boucle) ont

permis de dériver diverses versions d'algorithmes et ce, dans les cas de stockage dense et compressé de la matrice creuse traitée. Des tests expérimentaux sur divers graphes ont validé notre étude et permis de retenir les meilleures versions.

Comme perspectives de notre travail, nous retenons les points suivants : (i) expérimentations sur des graphes de plus grande taille afin de mieux confirmer les performances des deux meilleurs algorithmes conçus, (ii) prise en compte de l'aspect creux de la matrice C, (iii) parallélisation des versions séquentielles les mieux adaptées.

6. Bibliographie

[1] A. V. Aho, M. S. Lam, R. Sethi & J. D. Ullman, *Compilers: Principles, Techniques, & Tools*. (2nd edition), Pearson Addison Wesley, 2007.

[2] A.J.C. Bik, Compiler support for sparse matrix computations, Thèse, University of Leiden, The Netherlands, 1996.

[3] B. Bollobas, *Modern graph theory*, Springer, New York, 1998.

[4] C. Berge, *Graphes et hypergraphes*, Dunod, Paris, 1970.

[5] M. Cosnard & D. Trystram, *Algorithmes et architectures parallèles*, InterEditions, Paris, 1993.

[6] K.K. Damian, B. Comm and M. O'Neill Garret, The minimum Cost Flow Problem and The Network Simplex Method, Dissertation de Mastère, Université College Gublin, Irlande, 1991.

[7] T.A. Davis, *Direct methods for sparse linear systems*, SIAM, Philadelphia, 2006.

[8] P. M. Gleiss. Short Cycles, Minimum Cycle Bases of Graphs from Chemistry and Biochemistry, Thèse, Faculté des Sciences naturelles et Mathématiques, Wien, 2001.

[9] H. Hasni, H. Guesmi & Z. Mahjoub, Decomposition, size reduction and alternative formulations of separable minimum cost network flow problem, *à soumettre à RAIRO-RO*, 2011.

[10] O. Hamdi-Larbi, Étude de la distribution sur système à grande échelle de Calcul Numérique Traitant des Matrices Creuses Compressées, Thèse de Doctorat, Université de Tunis El Manar, Faculté des Sciences de Tunis, 2010.

[11] D. Klingman, A. Napier and J. Stutz, *NETGEN: A Program for Generating Large Scale Capacitated Assignment, Transportation, and Minimum Cost Flow Network Problems*, Management Science, *(*20), 814-821, 1974.

[12] M. Luong, D. Maquin & J. Ragot, Sensor Network Design for Detection and Isolation, *IFAC Conference on Control of Industrial Systems*, Belfort, France, 1997.

[13] V. Loechner, B. Meister & P. Clauss, Precise Data Locality Optimization of Nested Loops, *The Journal of Supercomputing*, 21(1), 37-76, 2002.