Constructing elimination trees for sparse unsymmetric matrices

Kamer Kaya, Bora Uçar

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Constructing elimination trees for sparse unsymmetric matrices

Kamer Kaya — Bora Uçar

N° 7549

February 2011

Distributed and High Performance Computing

ISSN 0249-6399 ISRN INRIA/RR--7549--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Constructing elimination trees for sparse unsymmetric matrices

Kamer Kaya* , Bora Uçar†

Theme : Distributed and High Performance Computing
Équipe-Projet GRAAL

Rapport de recherche n° 7549 — February 2011 — 7 pages

Abstract: The elimination tree model for unsymmetric matrices and an algorithm for constructing it have been recently proposed [Eisenstat and Liu, SIAM J. Matrix Anal. Appl., 26 (2005) and 29 (2008)]. The construction algorithm is of time complexity $\mathcal{O}(mn)$ for an $n \times n$ unsymmetric matrix having m nonzeros. We propose another algorithm with a time complexity of $\mathcal{O}(m \log n)$.

Key-words: Elimination tree, sparse matrix factorization

* CERFACS, Toulouse, France
† CNRS and ENS Lyon, France

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Construire les arbes d'élimination pour des matrices creuses non symétriques

Résumé : Eisenstat et Liu ont récemment décrit l'arbre d'élimination pour des matrices creuses non symétriques et ont proposé un algorithme pour le construire en temps $\mathcal{O}(mn)$ pour une matrice de taille n ayant m éléments non nuls [Eisenstat and Liu, SIAM J. Matrix Anal. Appl., 26 (2005) and 29 (2008)]. Nous décrivons un algorithme dont la complexité en temps est $\mathcal{O}(m \log n)$.

Mots-clés : Arbre d'élimination, factorisation des matrices creuses

1 Introduction

Arguably, the elimination tree is the single most important data structure in sparse matrix factorization methods. It is used to estimate and optimize the storage and computational requirements during symbolic and numerical factorizations [4]. Although the elimination tree for symmetric (positive definite) systems goes back to 80s [5] and even before (see [1, Section 3.3] and [4]), the elimination tree for unsymmetric matrices is a recent one. Eisenstat and Liu [2] define the the elimination tree for unsymmetric matrices and discuss its properties. In a follow up paper [3], they describe algorithms to construct those trees, to reorder matrices into a special form, and to use the trees to perform symbolic factorization. In this short note, we present an $\mathcal{O}(m \log n)$ time algorithm to construct the elimination tree of an $n \times n$ unsymmetric matrix with m nonzeros. The algorithm by Eisenstat and Liu [3] has a time complexity of $\mathcal{O}(mn)$.

Let $\mathbf{A}$ be a sparse $n \times n$ matrix $\mathbf{A}$ with m nonzeros, and a_{ij} be the entry at its ith row and jth column. The directed graph $G(\mathbf{A}) = (V, E)$ of $\mathbf{A}$ contains n vertices, $V = \{1, 2, \ldots, n\}$, and m edges, $E = \{ij : a_{ij} \neq 0, 1 \leq i, j \leq n\}$. For our purposes in this paper, we assume that $\mathbf{A}$ has a zero-free diagonal.

Let $G = (V, E)$ be a directed graph (digraph) where V and E are the sets of n vertices and m edges, respectively. An edge from vertex u to vertex v will be denoted as uv. If $u = v$, we say that uv is a *loop*. A *path* is an alternating sequence of vertices and edges which starts and ends with a vertex. If a path's first and last vertex are the same we say that path is *closed*. For two vertices $u, v \in V$, we say that u is *connected* to v in G, i.e., $u \overset{G}{\rightarrow} v$, if there is path from u to v.

A directed graph $G' = (V', E')$ is a *subgraph* of a larger digraph $G = (V, E)$ if $V' \subset V$ and $E' \subseteq E \cap (V' \times V')$. A subgraph is said to be a *maximal subgraph* if $E' = E \cap (V' \times V')$. Let $V = \{1, 2, \ldots, n\}$ and we have a well defined order on the vertices. Then, G_k denotes a special maximal subgraph containing the first k vertices in V. That is,

$$G_k = (\{1, \ldots, k\}, E \cap \{1, \ldots, k\} \times \{1, \ldots, k\}) .$$

If $u \overset{G}{\rightarrow} v$ for all $u, v \in V$, the digraph G is called *strongly connected*. If G' is *maximally strongly connected*, i.e., if there is no strongly connected subgraph G'' of G such that G' is a subgraph of G'', it is called a strong component (or a strongly connected component) of G. Due to the zero-free diagonal assumption, the vertices have self loops. Hence, if G contains a single vertex, it is strongly connected. A directed graph without cycles is called a Dag. The strongly connected components of a Dag are its vertices, i.e., each vertex forms a trivial strong component.

Let $\mathbf{A} = \mathbf{LU}$ be a nonsingular, sparse, unsymmetric $n \times n$ matrix with a nonzero diagonal where $\mathbf{L}$ and $\mathbf{U}$ be its lower and upper triangular factors, respectively. The *elimination tree* $T(\mathbf{A})$ of an unsymmetric $\mathbf{A}$ is defined by Eisenstat and Liu [2, 3] as follows.

$$\textsc{Parent}(i) = \{j : j > i \text{ and } j \overset{G(\mathbf{L})}{\rightarrow} i \overset{G(\mathbf{U})}{\rightarrow} j\} \tag{1}$$

where $\textsc{Parent}(i)$ shows the parent of vertex i in the elimination tree if i is not a root. Otherwise, $\textsc{Parent}(i) = \infty$. Eisenstat and Liu prove the following the-

orem in order to develop an algorithm for constructing $T(\mathbf{A})$ without knowing $G(\mathbf{L})$ and $G(\mathbf{U})$.

Theorem 1.1 (Theorem 3.3 of [2]) *Vertex k is the parent of vertex i in the elimination tree $T(\mathbf{A})$ if and only if k is the first vertex after i such that k and i belong to the same strongly connected component of the subgraph $G_k(\mathbf{A})$ of $G(\mathbf{A})$.*

Given this theorem, Eisenstat and Liu [3] propose the following algorithm to compute the parent pointers in (1) and construct the elimination tree.

Algorithm 1 ETREE

```
1: for vertex k = 1 to n do
2:    Find the component C of G_k(A) that contains k
3:    for each vertex i ∈ C \ {k} do
4:       if PARENT(i) = ∞ then
5:          PARENT(i) = k
6:    PARENT(k) = ∞
```

The strong components of a digraph with m edges and n vertices can be found in $\mathcal{O}(n+m)$ time [6]. Hence, the complexity of the algorithm ETREE is $\mathcal{O}(mn)$ for a graph containing n vertices and m edges. Eisentat and Liu reduce the practical running time of the algorithm by using quotient graphs [3] and specialized algorithms to find the strong component containing a given vertex, but report that the time complexity of the improved algorithm is still $\mathcal{O}(mn)$.

2 An $\mathcal{O}(m \log n)$ time algorithm

Let $\mathbf{A}$ be an irreducible, unsymmetric, $n \times n$ matrix with a nonzero diagonal. Algorithm 2 shows the proposed algorithm ETREESC which finds the elimination tree $T(\mathbf{A})$ in a bottom-up fashion. There are two inputs for the algorithm: a strongly connected digraph G with η vertices and an integer $s < \eta$ such that G_s is acyclic. That is, all the strong components of G_s are trivial.The algorithm initial call is ETREESC$(G(\mathbf{A}) = (\{1, 2, \ldots, n\}, E), 0)$. We assume that the parent pointers are initialized to ∞ before the execution of the algorithm. Our algorithm and its analysis are based on an algorithm by Tarjan [7].

For each call of ETREESC, each vertex $i_j \in V$ such that $j \leq s$ represents a subtree in $T(\mathbf{A})$ with root i_j. The algorithm tries to find the first vertex i_k that connects some or all of these subtrees. Since G is strongly connected and G_s is known to be acyclic, if $s = \eta - 1$, i_η must be the vertex that makes G strongly connected. If this is the case, we set the parent pointers appropriately. If this is not the case, the algorithm continues to the search with the vertex i_k where $k = \lceil (s + \eta)/2 \rceil$. If G_k is strongly connected, the search continues recursively. On the other hand, if G_k is not strongly connected, we know that each strong component corresponds to a subtree in the elimination tree. In this case, each recursive call at line 15 is responsible for setting the parent pointers of the vertices in these subtrees, and the one at 20 is responsible for setting the pointers of the roots of these subtrees. For the last call, we create a new graph $G' = (V', E')$ (from G), which contains a super-vertex for each strong

Algorithm 2 $r = \text{eTreeSC}(G = (\{i_1, i_2, \ldots, i_\eta\}, E), s)$

Output r: the root of the elimination tree of G

```
1:  if s = η − 1 then
2:    for j = 1 to η − 1 do
3:      Parent(i_j) = i_η
4:    return i_η        ▶ i_η is the root
5:  else
6:    k = ⌈(s + η)/2⌉
7:    Let p be the number of strong components of G_k
8:    if p = 1 then
9:      return eTreeSC(G_k, s)
10:   else
11:     for ℓ = 1 to p do
12:       Let C_ℓ = (V_ℓ, E_ℓ) be the ℓth strong component of G_k
13:       if |V_ℓ| > 1 then
14:         s_ℓ = |{i_j : i_j ∈ V_ℓ, j ≤ s}|     ▶ the first s vertices form a Dag
15:         r_ℓ ← eTreeSC(C_ℓ, s_ℓ)     ▶ find the root of each str. comp.
16:       else
17:         r_ℓ ← the vertex in V_ℓ
18:       Shrunk V_ℓ into the vertex r_ℓ
19:     Let V' = {r_1, ..., r_p, i_{k+1}, i_{k+2}, ..., i_η}
20:     return eTreeSC(G' = (V', E'), p)     ▶ G'_p is acyclic
```

component of G_k. That is, the vertex set V_ℓ of each strong component C_ℓ is shrunk into a single super-vertex $r_\ell \in V'$ for $1 \leq \ell \leq p$. In the elimination tree, r_ℓ is the root of the subtree containing the vertices in V_ℓ. The edge set E' contains an edge for each edge in E except the ones whose endpoints are in the same strong component of G_k, and a self loop for each super-vertex. If an endpoint of an edge is in a strong component of G_k, we use the corresponding super-vertex in V' for that edge.

For a recursive call with a strong component C_ℓ, s_ℓ is equal to the number of vertices in V_ℓ at an index smaller than or equal to s. Hence, the subgraph of C_ℓ containing the first s_ℓ vertices is acyclic. On the other hand, for the graph G', we know that the first p vertices form a directed acyclic graph since they correspond to the strong components of G_k.

We adapt the analysis in [7] to find the time complexity of Algorithm 2. Forgetting the recursive calls, the complexity of the body of eTreeSC is equal to the complexity of assigning the parent pointers and finding the strong components which is $\mathcal{O}(m)$. In total, line 3 will be executed as many times as the number of vertices in the original graph. For the recursive calls, each edge of G is used in at most one recursive call. Let $\eta - s$ be the *rank* of the problem, i.e., the number of vertices to be searched. The ranks of the recursive calls at lines 9, 15 and 20 are at most $\max(k - s, \eta - k)$. As k, s and η are integers, and due to the definition of k at line 6, for a call with rank ρ, the ranks of its recursive calls are at most $2\rho/3$. Let $R(m, \rho)$ be the time complexity of a problem (excluding the part for setting the parent pointers) with m edges and

rank ρ, and c be the number of recursive calls. Then

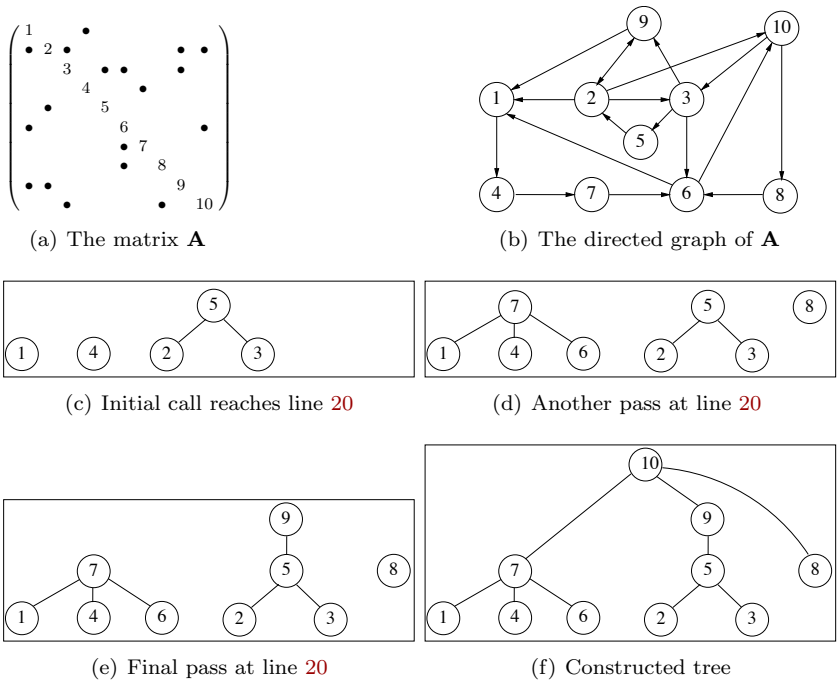
$$R(m,\rho) = \mathcal{O}(m) + \sum_{i=1}^{c} R(m_i,\rho_i).$$

By induction, we will show that $R(m,\rho) = \mathcal{O}(m\log\rho)$. We assume that the equation is correct for all m_i and ρ_i. For the base case, when $\rho = 1$, the algorithm only sets the parent pointers hence the assumption holds. Since, $\sum_{i=1}^{c} m_i \le m$, by the assumption,

$$R(m,\rho) = \mathcal{O}(m) + R\left(m, \frac{2\rho}{3}\right) = \mathcal{O}(m\log\rho)$$

where the last equality is due to the well-known master theorem. Hence, the time complexity of the algorithm is $\mathcal{O}(m\log n + n)$. As $m \ge n$, we can conclude that the complexity is $\mathcal{O}(m\log n)$.

Consider the sample matrix from [3] shown in Fig. 1(a) whose graph is shown in Fig. 1(b). The first call with the whole graph finds $k = 5, p = 3$ with $V_1 = \{1\}$,



(a) The matrix **A**

(b) The directed graph of **A**

(c) Initial call reaches line 20

(d) Another pass at line 20

(e) Final pass at line 20

(f) Constructed tree

Figure 1: Tracing the algorithm on the sample matrix.

$V_2 = \{4\}$, $V_3 = \{2,3,5\}$, and then recursively calls the algorithm in V_3; then the algorithm reaches to line 20, yielding the subtree shown in Fig. 1(c) with $V' = \{r_1 = 1; r_2 = 4; r_3 = 5; 6; 7; 8; 9; 10\}$. The recursive call is invoked with $s = 3$ and then finds $k = 6$ (pointing to vertex 8). Three strong components $V_1 = \{1,4,6,7\}$, $V_2 = \{2,3,5\}$, and $V_3 = \{8\}$ are found, and 7 becomes the root of V_1 after the recursive call (the subtree is at Fig. 1(d)). Then the algorithm reaches again to line 20 with $V' = \{r_1 = 7; r_2 = 5; r_3 = 8; 9; 10\}$ and $s = 3$.

This time $k = 4$ (points at vertex 9). Again three strong components are found and 9 becomes the root of the component containing 5, yielding the tree shown in Fig. 20. The last call results in the elimination tree shown in Fig. 1(f).

3 Conclusion

We have proposed an algorithm to compute the elimination tree of an $n \times n$ unsymmetric matrix with m nonzeros in $\mathcal{O}(m \log n)$ time. The previously known algorithm [3] is of time complexity $\mathcal{O}(mn)$.

The proposed algorithm is based on an algorithm by Tarjan [7]. It is interesting to note (as noted by Liu [4, p.142]) that the elimination tree for symmetric matrices can also be constructed by an algorithm by Tarjan and Yannakakis (the fill-in computation algorithm [8, p. 570]). If one removes the line "add $\{x, v\}$ to" and changes the test "$index(x) < i$" into"$f(x) \neq x$" then the $f(\cdot)$ pointers become exactly the parent pointers. The modifications we needed for the unsymmetric case are a little more involved than those for the symmetric case, but still this is a curious coincidence.

References

[1] Iain S. Duff and Bora Uçar. Combinatorial problems in solving linear systems. Technical Report TR/PA/09/60, CERFACS, Toulouse, France, 2009.

[2] Stanley C. Eisenstat and Joseph W. H. Liu. The theory of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 26:686–705, January 2005.

[3] Stanley C. Eisenstat and Joseph W. H. Liu. Algorithmic aspects of elimination trees for sparse unsymmetric matrices. *SIAM J. Matrix Anal. Appl.*, 29:1363–1381, January 2008.

[4] Joseph W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[5] Rob Schreiber. A new implementation of sparse Gaussian elimination. *ACM T. Math. Software*, 8(3):256–276, 1982.

[6] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[7] Robert E. Tarjan. An improved algorithm for hierarchical clustering using strong components. *Inform. Process. Lett.*, 17(1):37–41, 1983.

[8] Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399