Fast computation of minimal fill inside a given elimination ordering

Pinar Heggernes* Barry W. Peyton†

Abstract

Minimal elimination orderings were introduced by Rose, Tarjan, and Lueker in 1976, and during the last decade they have received increasing attention. Such orderings have important applications in several different fields, and they were first studied in connection to minimizing fill in sparse matrix computations. Rather than computing any minimal ordering, which might result in fill that is far from minimum, it is more desirable for practical applications to start from an ordering produced by a fill-reducing heuristic, and then compute a minimal fill that is a subset of the fill produced by the given heuristic. This problem has been addressed previously, and there are several algorithms for solving it. The drawback of these algorithms is that either there is no theoretical bound given on their running time although they might run fast in practice, or they have a good theoretical running time but they have never been implemented, or they require a large machinery of complicated data structures to achieve the good theoretical time bound [2]. In this paper, we present an algorithm called MCS-ETree for solving the mentioned problem in $O(nm\,\mathrm{A}(m,n))$ time, where m and n are respectively the number of edges and vertices of the graph corresponding to the input sparse matrix, and $\mathrm{A}(m,n)$ is the very slowly growing inverse of Ackerman's function.

A primary strength of our algorithm is its simplicity and its straightforward implementation details. MCS-ETree numbers the vertices from n down to 1. The unnumbered vertices are stored in elimination subtrees, one elimination subtree for each connected component remaining to be numbered. These elimination subtrees serve to preserve the qualities of the initial fill-reducing ordering, and to record the necessary changes to that ordering. Let $T[v]$ be an elimination subtree for one of these unnumbered connected components, where v is the root.

At each step, the algorithm selects an unnumbered elimination subtree $T[v]$. The key feature is the selection at each step of an unnumbered vertex u in $T[v]$ of "maximum cardinality" to give the next number to. Not any vertex of maximum cardinality can be selected; it has to be a vertex of maximum cardinality in $T[v]$ that has no descendants of maximum cardinality. This is the feature of the algorithm that causes it to produce a minimal ordering. To implement the selection of maximum cardinality vertices in $O(nm\,\mathrm{A}(m,n))$ total time, we adapt for our purposes the algorithm in [1] for computing counts of column nonzeros in a Cholesky factor.

After selection of maximum cardinality vertex u, the algorithm must compute an reordering of $T[v]$ that produces equivalent fill but numbers u last. Reordering in this way preserves

*Department of Informatics, University of Bergen, N-5020 Bergen, Norway. Email: pinar@ii.uib.no
†Dalton State College, 650 College Drive, Dalton GA 30720, USA. Email: bpeyton@daltonstate.edu

the fill-reducing quality of the initial ordering. To implement computation of the equivalent reorderings in $O(mn)$ total time, we use a trivial modification of Composite_Rotations in [3].

After this equivalent reordering, the new elimination subtree $T'[u]$ rooted at maximum cardinality vertex u must be computed. Total time for computation of elimination subtrees is $O(nm\,\mathrm{A}(m,n))$ [4]. At this point, the algorithm gives maximum cardinality vertex u its number in the final ordering, and stores each elimination subtree $T'[c]$, where c is a child of u, for subsequent processing.

We coded a `basic` implementation of MCS-ETree, where we use the most straighforward implementation of the tools cited above. This implementation traverses the adjacency lists three times for every vertex in each elimination subtree $T[v]$ that is processed. But it is straighforward to enhance the implementation so that it avoids much of the processing of adjacency lists for vertices that are not among the ancestors of the maximum cardinality vertex u chosen in $T[v]$. We coded an `enhanced` version based on these improvements. On top of these enhancements, we incorporated a straightforward technique based on indistinguishable vertices that enables MCS-ETree to number a block of vertices during a step. We coded a `blocked` version of MCS-ETree based on this additional improvement.

For our run time tests, initial orderings were obtained from Approximate Minimum Degree (AMD); we also tested some with random initial orderings. Our run time tests show that the `basic` implementation runs quite slowly, as expected. The tests show that the `enhanced` implementation runs much faster than the `basic` implementation, but it is still too slow for practical sparse matrix computations. The `blocked` implementation runs much faster than the `enhanced` implementation. Using AMD initial orderings, the `blocked` implementation of our algorithm is fast enough to be considered for sparse matrix computations.

To summarize, our algorithm for this problem is the first that both has a provably good running time with straightforward implementation details, and is fast in practice.

References

[1] J. R. Gilbert, E. G. Ng, and B. W. Peyton, *An efficient algorithm to compute row and column counts for sparse Cholesky factorization*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1075–1091.

[2] P. Heggernes, *Minimal triangulations of graphs: A survey*, Disc. Math., 306 (2006), pp. 297–317.

[3] J. W. H. Liu, *Equivalent sparse matrix reorderings by elimination tree rotations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 424–444.

[4] ——, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 134–172.