A Detailed Comparison of Symmetric Sparse Matrix Reordering Heuristics

Charles Hornbaker
Institute for Applied Computational Science
Harvard University
Cambridge, MA 02138
chornbaker@fas.harvard.edu

Anita Mehrotra
Institute for Applied Computational Science
Harvard University
Cambridge, MA 02138
anitamehrotra@fas.harvard.edu

Jody Schechter
Institute for Applied Computational Science
Harvard University
Cambridge, MA 02138
schechter@fas.harvard.edu

Abstract

Sparse symmetric graphs dominate current research and represent a rapidly growing area of interest in industry, ranging from social networks to condensed matter physics. A set of reordering heuristics exist to counteract the fill-in resulting from Cholesky decomposition of these matrices [1-5]. In this paper, we compare four reordering algorithms for random and special symmetric, sparse, positive-definite matrices, then measure the cost of each algorithm using six distinct metrics. In an effort to measure computational accuracy, we then recompute the reordered matrix from the Cholesky decomposition and measure the relative error. We find, on average, CM and RCM reorder random matrices most quickly, Minimum Degree has the largest reduction in fill-in, and King's has the largest change in profile. We conclude that, consistent with current literature, the "ideal" reordering algorithm depends largely on the structure of the matrix.

1 Introduction

Symmetric, sparse matrices can be used to represent undirected graphs. As a result, such matrices play a critical role in graph theory, both within the realm of numerical analysis research and also in fields as varied as sociology and biology. However, algorithms that operate on such matrices often depend on the classic Cholesky decomposition. The Cholesky decomposition results in an upper-triangular matrix L, which, when multiplied with its' transpose L^T, results in a close approximation to the original matrix A. This decomposition is used often specifically because it can reduce runtimes for common operations.

The amount of fill-in introduced during Cholesky decomposition is dependent on the ordering of the sparse symmetric matrix. Finding the optimal ordering to minimize fill-in is an NP-hard problem. Hence, heuristic methods have been developed to reorder sparse matrices, with an eye towards reducing Cholesky fill-in.

We provide a comprehensive overview of four heuristic methods for reordering matrices. Then, we evaluate these methods using six cost metrics: time to reorder, bandwidth reduction, profile reduction, reduction in Cholesky decomposition time, change in fill-in, and change in the height of the elimination tree.

2 Background

2.1 Literature Overview

Many heuristics have been proposed to reorder sparse symmetric matrices. Markowitz (1957) proposed a Minimum Degree algorithm, also known as the "greedy algorithm," which selects the node of minimum degree to eliminate at each step of the elimination game [11]. Two variants have been proposed since: the Multiple Minimum Degree algorithm (MMD) and the Approximate Minimum Degree algorithm (AMD), which have been shown to reduce runtime [6]. Cuthill and McKee proposed another method in 1969, which primarily aims to reduce bandwidth of the resulting matrix [1]. George (1971) showed that reversing the Cuthill-McKee algorithm reduces fill-in of the resulting Cholesky decomposition even further [3]. King (1970) made an alternative modification, focused on profile reduction [10]. In 1976, Gibbs, Pool, and Stockmeyer [4] published results from a new alogrithm that they claim reduces computation time while offering similar reduction of bandwidth of the sparse matrices to the algorithm proposed by Cuthill and McKee [1]. More recently, Kaveh and Sharafi [9] proposed an Ant algorithm, which they claim outperforms even Gibbs, Pool, and Stockmeyer.

In this paper, we implement four of the major algorithms listed above, and evaluate them based on six different cost measures. Our work is informed not only by the descriptions of the algorithms included in the literature, but also by the parameters of evaluation used in previous research. We evaluate different heuristics on a uniform set of cost measures that we believe, based on our review of the literature, effectively demonstrate the utility of each algorithm.

2.2 Graphs and Fill-in

Figure 1 shows an undirected graph and its corresponding symmetric adjacency matrix, where each non-zero element in column i and row j denotes an edge between vertices i and j. In the first step of the Cholesky process, the vertex corresponding to the first row and column is removed from the graph, and edges are drown among its neighbors. "Fill-in" refers to the new non-zero entries in the matrix that correspond to the new edges. As you can see, in this toy-example, the first elimination creates twelve new non-zero entries, denoted in red in the graph. For small matrices, the cost of computation and expense of storage space in negligible, but for large, sparse matrices, a great deal of fill-in can occur during Cholesky decomposition.

The question that reordering algorithms seek to address is, in which order should we eliminate the vertices from the elimination graph to minimize fill-in, while keeping computational costs low?

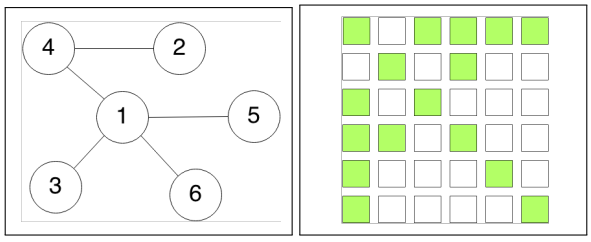

Figure 1: A depiction of a graph and its adjacency matrix.

3 Reordering Algorithms

3.1 Minimum Degree

The minimum degree algorithm finds the node of minimum degree, then eliminates it from the graph. It then updates the other nodes' connections given that that node was eliminated, before beginning again in search of the new node of minimum degree.

In our implementation of the algorithm, we used a quotient graph with "supernodes" (also known as the "snode") composed of indistinguishable nodes. Two nodes are considered to be indistinguishable

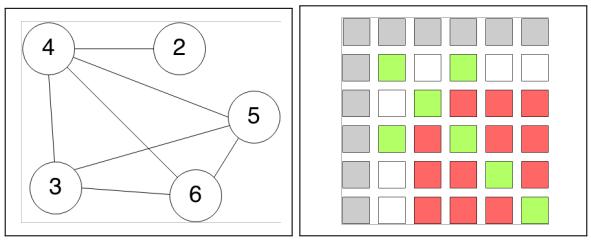

Figure 2: Graph and adjacency matrix from Figure 1, after removing a node.

if their adjacency sets are the same. A supernode is weighted by the number of indistinguishable nodes contained within it. Where a node is distinguishable from all other nodes in the set, it is considered to be a supernode with weight one.

1. Eliminate the nodes of minimum degree by updating them from snodes to "enode", meaning eliminated node.
2. Find a "reachable" set of the supernode we remove, which includes all of the snodes in its adjacency set and all of the snodes connected to enodes in its adjacency set.
3. Update the degree of every snode in the reachable set to be the sum of the weights of snodes in *their* reachable sets, before finding the next snode of minimum degree.

The Minimum Degree algorithm is known to be computationally expensive, and there are two popular variants that aim to reduce runtime. One variant is the **multiple minimum degree** (MMD) algorithm. In this version, nodes that are of the same minimum degree, but are independent of one another – that is, are not in one another's adjacency sets – are eliminated at once.

Another variant is the **approximate minimum degree** (AMD) algorithm. Like in the original minimum degree algorithm, in this heuristic method, snodes are eliminated one-at-a-time, but rather than calculating the exact degrees of the snodes in the reachable set, this algorithm approximates the degree of an snode r to be the combined weights of all the snodes and enodes in the adjacency set of r. Thus, to update the degree of an snode r in the reachable set of the eliminated node e, this algorithm adds the weight of e to the weight of r and subtracts the weight of r, since the weight of e should account for the weights of all of the other supernodes to which r is connected through e.

We choose to implement the original minimum degree algorithm, knowing that its runtime will likely be slower than the approximate and multiple minimum degree versions. This algorithm, as opposed to AMD, gives accurate degree measurements at each step.

3.2 Cuthill-McKee

The Cuthill-McKee algorithm (CM) provides a method for systematically renumbering the rows and columns of a symmetric matrix so as to reduce the bandwidth with the resulting permutation. This algorithm is a variation of a breadth-first search, where nodes are numbered in order of increasing degree. CM can be implemented in $O(q_{max}m)$ running time, where m is the number of edges and q_{max} is the maximum degree of any node. In sparse networks, this becomes $O(n)$, where n is the number of nodes.

The basic algorithm is:

1. Select a node of minimum degree as the starting node and add it to the permutation vector.
2. For $i = 1, 2, ...n$, sort the nodes adjacent to node i in order of increasing degree, ignoring any nodes that have already been considered, and add them to the permutation vector.
3. Repeat (2) until all nodes have been processed. If node i has no unlabeled adjacencies, select another node of minimum degree and resume the process. This accounts for matrices with multiple components [1, 12].

We implement the CM algorithm in MatLab as described above. Our implementation performs reasonably well on matrices up to approximately 1×10^4, but exhibits significant performance loss with larger matrices. This is likely associated with unoptimized implementations such as the use of sorting functions within the main loop.

3.3 Reverse Cuthill-McKee

The Reverse Cuthill-McKee (RCM) is identical to the CM, but includes an additional step of reversing the order of the permutation vector at the end of the algorithm. In [3], George shows that RCM reduces the profile in most cases, while leaving the bandwidth unchanged. Because there is little additional computational cost for the final step to gain the additional profile benefit, RCM is commonly used instead of CM.

In our implementation of RCM, the node with lowest degree is the starting node. Current literature suggests alternative methods for choosing the starting node, such as using a pseudo-peripheral node, in order to improve performance.

3.4 King's Algorithm

King's algorithm is another variant of the RCM that reduces the profile of the permuted matrix. Unlike RCM, King's algorithm orders nodes based on the number of connections they have to already processed vertices rather than based on their total degree [9]. The basic algorithm is as follows:

> 1. Select a node of minimum degree as the starting node and add it to the permutation vector p.
> 2. For $i = 1, 2, ...n$, Find the unprocessed adjacencies of the nodes in p and add the node that causes the smallest subset of the remaining unprocessed nodes to be added.
> 3. Repeat (2) until all nodes have been processed. If node i has no unlabeled adjacencies, select another node of minimum degree and resume the process. This accounts for matrices with multiple components.

The primary difference in King's algorithm is in step (2), where only one node at a time is added to p. This allows a node that may be part of a local cluster to be ordered in close proximity to the other nodes in the cluster. The result is a lower profile at the cost of higher bandwidth.

4 Methodology

4.1 Evaluation of Random and Special Matrices

The first portion of our project focuses on understanding and implementing the Minimum Degree, CM, RCM, and King algorithms. We next evaluate the costs and benefits of each algorithm.

We perform our evaluation in two steps: by testing on random matrices and by testing on special matrices. With random matrices, we expect lower-levels of connectivity and therefore low bandwidth after reordering. However, this allows us to scale in matrix size in order to test our algorithms on very large matrices. Testing on special matrices from real-world applications allows us to then validate our metrics on a more highly-connected, structurally-dependent sparse graphs.

First, we generate random matrices that are sparse, symmetric and positive-definite, with elements drawn from a standard Gaussian distribution. We apply the reordering algorithms to ten random matrices of size 10^N and density 10^{-N}. We then repeat this process for all values of N where $N \in \{1, 2, 3, 4\}$.

Next, we test our algorithms on a set of three special matrices, obtained from the University of Florida's Sparse Matrix Collection [1, 7, 17]:

- Mesh2EM5, size = 306×306. The Mesh2EM5 is an example of a structural problem graph with 2D structure. It was obtained from the Pothen group at NASA.

- MSC01050, size = 1050×1050. The MSC01050 matrix is a structural engineering matrix from Boeing and the MacNeal Schwendler Corporation (MSC) at NASTRAN, a finite element analysis program developed for NASA that specializes in simulation software.
- STS4098, size = 4098×4098. The STS4098 from the Cannizzo group also represents a finite-element structural engineering matrix.

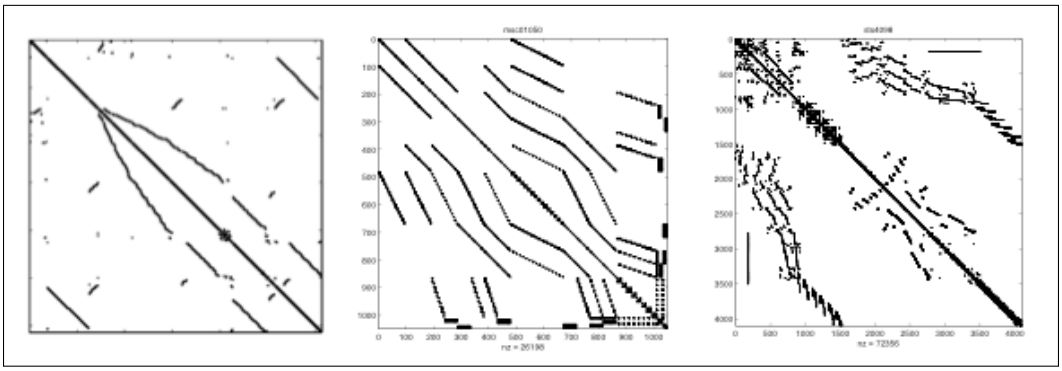

Figure 3: Sparsity patterns of special matrices (Mesh2EM5, MSC01050 and STS4098, respectively).

All tests were performed on a 2013 MacBook Pro, OSX version $10.8.5$, 1.3 GHz Intel Core i5 with 8 GB of memory and 1600 MHz DDR3.

To measure the cost associated with reordering, we implement six distinct cost metrics.

1. Time to reorder matrix
2. Time to run Cholesky decomposition
3. Fill-in after applying the Cholesky decomposition
4. Height of the elimination tree
5. Bandwidth
6. Profile

Matrix Reordering Runtime

We measure the average time (T) it takes to reorder matrices (either random and of size 10^N, or one of the special matrices described above), using each of the four algorithms. The biggest drawback to using this as the sole cost measurement, is that the time to execute a reordering is dependent on several variables: the computer it is run on and its hardware, the implementation of the algorithm, optimization techniques used, etc.

Cholesky Decomposition Runtime

We also measure the average amount of time it takes to run the Cholesky decomposition, both on the original matrix (T_{chol_0}) and on the reordered matrix (T_{chol_1}). Because of its numerically stable properties, the Cholesky decomposition is often applied to sparse matrices. We therefore time this decomposition as one of our cost metrics.

Cholesky Decomposition Fill-in

"Fill-in" is a measure that captures the number of zero entries that become non-zero after a transformation. For this metric, we compute the number of non-zero entries in a lower triangular portion of the original symmetric matrix. We then count the number of non-zeros in the upper-triangular matrix resulting from the Cholesky decomposition, as well as the number of zeros in the lower-triangular matrix resulting from the Cholesky decomposition of the *reordered* matrix. Subtracting the number of original non-zero elements from each of these gives us the fill-in. $\triangle\theta$ is the difference between these numbers and captures the change in fill-in.

Bandwidth

Bandwidth intuitively captures the maximum width of the middle band that can be seen in the sparsity pattern of the original and reordered matrices. Mathematically, it is the maximum of the differences between the row index and column index of the non-zero elements:

$$\text{bandwidth} = \max_{i \in I, j \in J} |i - j|,$$

where I and J are the row and column indices of all non-zero elements in the matrix.

Profile

The profile metric measures the total width across the banded middle of the sparsity pattern of a matrix (as opposed to the maximum width captured by bandwidth). It is computed using the following formula:

$$\text{profile} = \sum_{i=1}^{N} \delta_i, \text{ where } \delta_i = |i - f_i|,$$

and f_i is the index of the left-most non-zero element in the matrix.

Elimination Tree Height

Formally, the "elimination tree" of a Cholesky decomposition is a spanning tree of the filled-in graph satisfying the relation $PARENT(j) = \min_{\ell_{ij} \neq 0}\{i > j\}$, where ℓ_{ij} denotes a non-zero element. Intuitively, it is the height of the tree that captures the steps of transforming/reordering the original matrix to the current version.

4.2 Impact of Reordering on Computation

In addition to conserving storage space through bandwidth and profile reduction, reordering algorithms are expected to provide an advantage in computations using the permuted matrix. In our evaluation of the four algorithms, we are interested in seeing if the ordering has a predictable effect on the accuracy of basic computations.

In the second portion of our project, we measure the Frobenius norm error in the reconstructed matrix $\hat{A} = L \cdot L^T$ versus the original matrix A. The overall error is then calculated using the formula

$$\text{error} = ||A - \hat{A}||_{Frobenius}.$$

5 Results and Analysis

A figure demonstrating the flow of our key scripts can be found in the Appendix.

5.1 Evaluation of Random and Special Matrices

5.1.1 Comparison across Random Matrices

A comprehensive summary table of the results for random matrices can be found in the Appendix.

Figure 4 is a visual representation of the reorderings on a 300×300 sparse, symmetric, positive definite random matrix. Note the small bandwidth due to the low connectivity of random matrices:

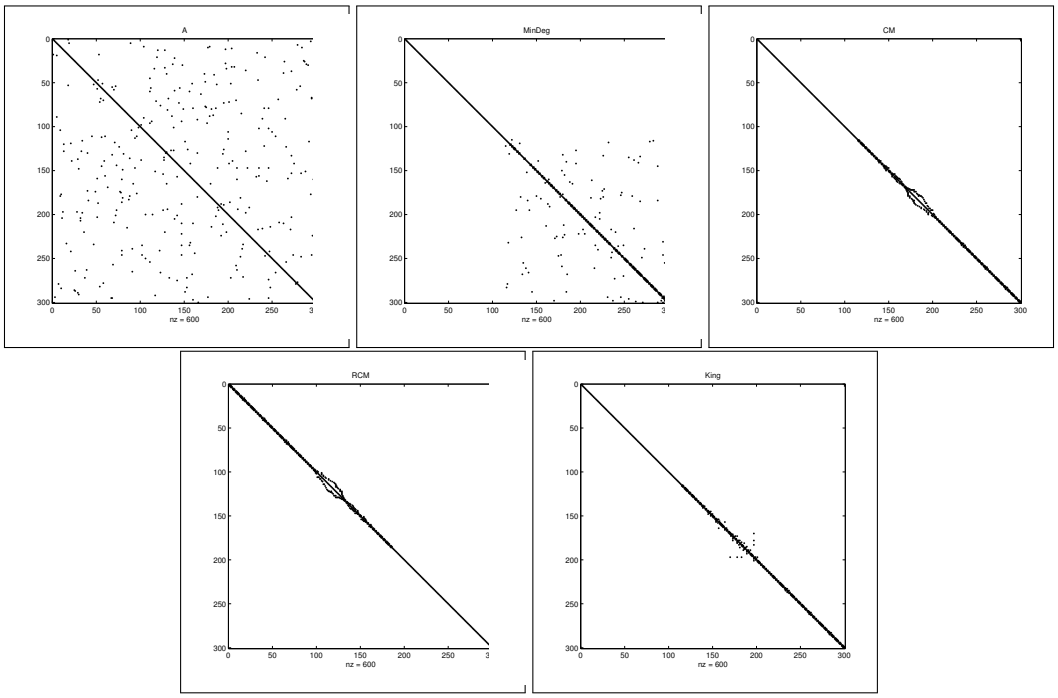

Figure 4: Sample random matrix, and applications of the reordering algorithms.

Matrix Reordering Runtime

Let n be the number of nodes in a graph and m be the number of adjacencies of all the nodes in the graph. In theory, the running time for the minimum degree algorithm is bounded above by $O(n^2m)$, and the running time for the RCM, CM, and King's algorithms should all be bounded above by $O(n^2)$ and below by $O(n)$. In the plot below, we show the timed results of runs of our algorithms on random sparse matrices with varying values of n and constant densities. For reference, we've also included lines of $O(n)$ and $O(n^2)$. The plot shows that are algorithms scale in accordance with theoretical predictions.

Based on the trend in this plot, we expect the runtime for a random matrix with $n = 10^5$ entries to be on the order of 10^5 seconds for the minimum degree algorithm. Indeed, we find that running the algorithms on matrices of this magnitude on matrices of this size – and then averaging across ten trials – is impractical on a single core.

A plot of the times per reordering algorithm, per matrix is shown below:

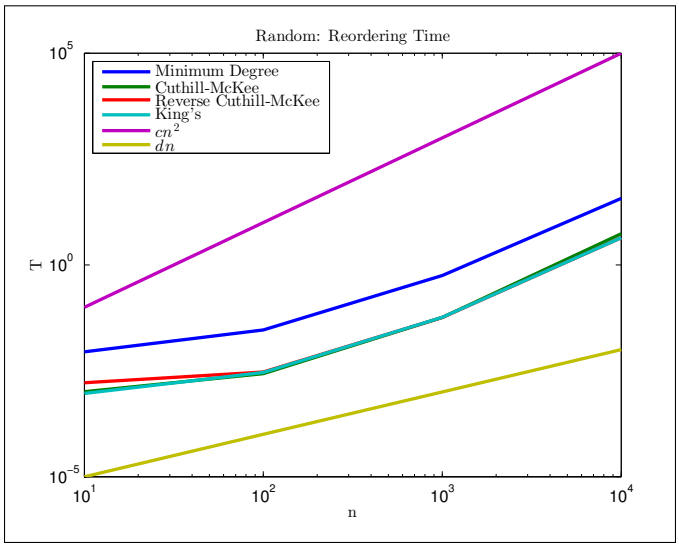

Figure 5: Time of reordering.

Cholesky Decomposition Runtime

We make Cholesky decomposition runtime comparisons between the original and reordered matrices using each of our four algorithms. The metric in the plot below is the speedup in the Cholesky decomposition of the reordered matrix, as compared to the original matrix. This is captured in the formula

$$\text{speedup} = \frac{T_{chol0}}{T_{chol1}}.$$

In the plot below, we include more points to better show the trends as matrix size grows. For matrices with dimensions of at least 10^2, the RCM algorithm yields matrices with the fastest decomposition times. Ironically, CM, which just gives the same matrix as the RCM algorithm in the opposite order, offers the least speedup of the four. This result has been observed in the literature and is to be expected.

A plot of the speed-ups per reordering algorithm, per matrix is shown below. It exhibits some inconsistencies, attributed to machine-specific computational resource availability. The trends, however, remain consistent over multiple trials.

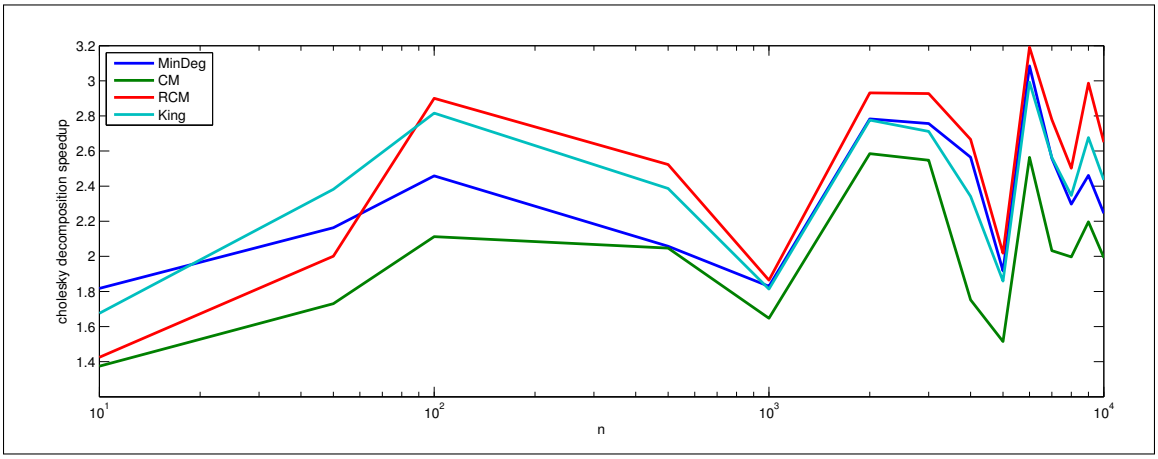

Figure 6: Speed-up.

We now turn to characteristics of the matrices themselves that we believe could help to explain this phenomenon.

Cholesky Decomposition Fill-in

First, we plot the average change in fill-in that occurs during the Cholesky decomposition of each heuristic reordering. The Cuthill-McKee algorithm returns the matrices that yield the most fill-in of the four, while the Reverse Cuthill-McKee returns the matrices that yield the least. This explains why reversing the rows has such a drastic impact on the decomposition time. Fill-in slows Cholesky down.

We think that it is interesting that Reverse Cuthill-McKee and Minimum Degree both yield matrices with the same amount of fill-in, given how visually different the reordered matrices look. Since RCM consistently has a faster Cholesky decomposition time than the Minimum Degree, we hypothesize that the shape of the reordered matrix influences decomposition time. We also note here that Cholesky decomposition of matrices reordered by the King's algorithm does not create much less fill-in than the original matrix.

A plot of the fill-in per reordering algorithm, per matrix is shown below:

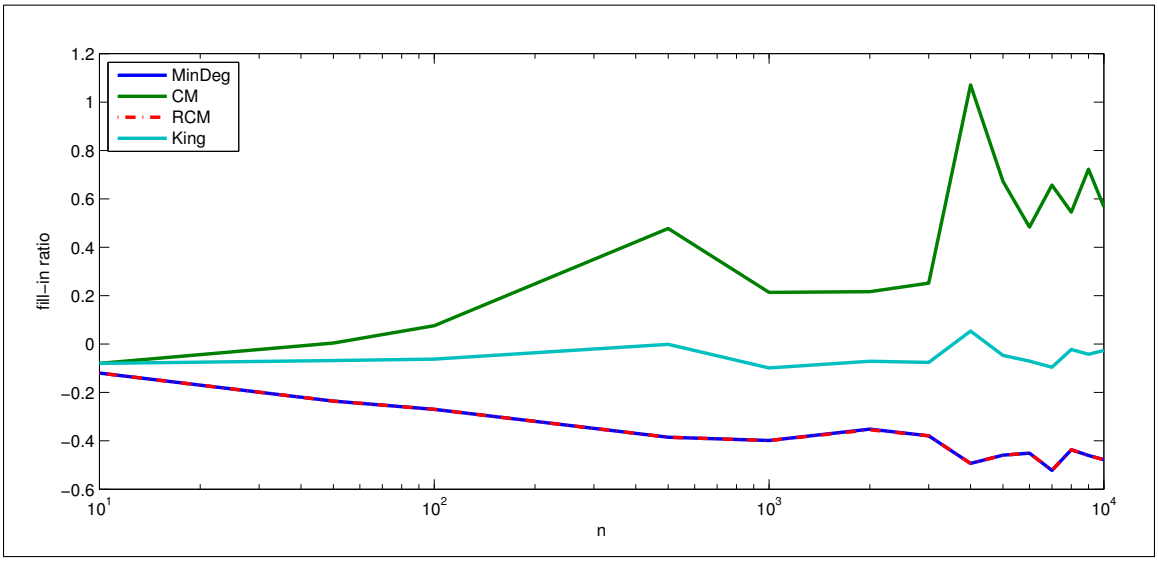

Figure 7: Fill-in.

Bandwidth and Profile

We examine shape in two ways: bandwidth and profile. The plots below show that the bandwidth and profile measurements for matrices reordered by all of King's, RCM, and CM are very close. They also show that the minimum degree algorithm tends to reduce bandwidth far less than the other algorithms, on average. This helps to explain why the RCM algorithm outpaces the Minimum Degree algorithm in terms of Cholesky decomposition time, despite the near-equal fill-in under Cholesky for each.

A plot of the bandwidth and profile per reordering algorithm, per matrix is shown below:

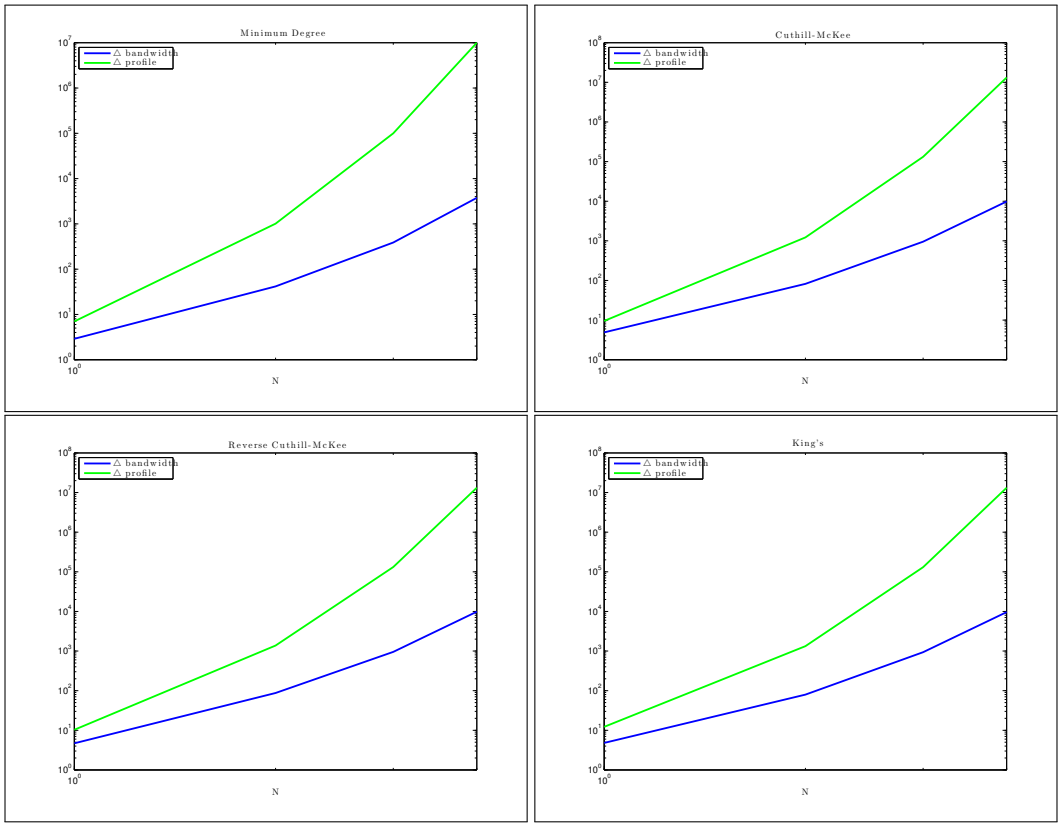

Figure 8: $profile$ and $bandwidth$.

Elimination Tree Height

Finally, we examine the height of the elimination tree as related to the Cholesky decomposition time. You can see in the plots below that the elimination trees for the Reverse Cuthill-McKee and Minimum Degree algorithm tend to be much shorter than those for the Cuthill-Mckee and King's algorithm. These vastly shorter elimination trees pay off in decomposition time.

A plot with a comparison of the height of elimination tree and Cholesky decomposition time per reordering algorithm, per matrix is shown below:

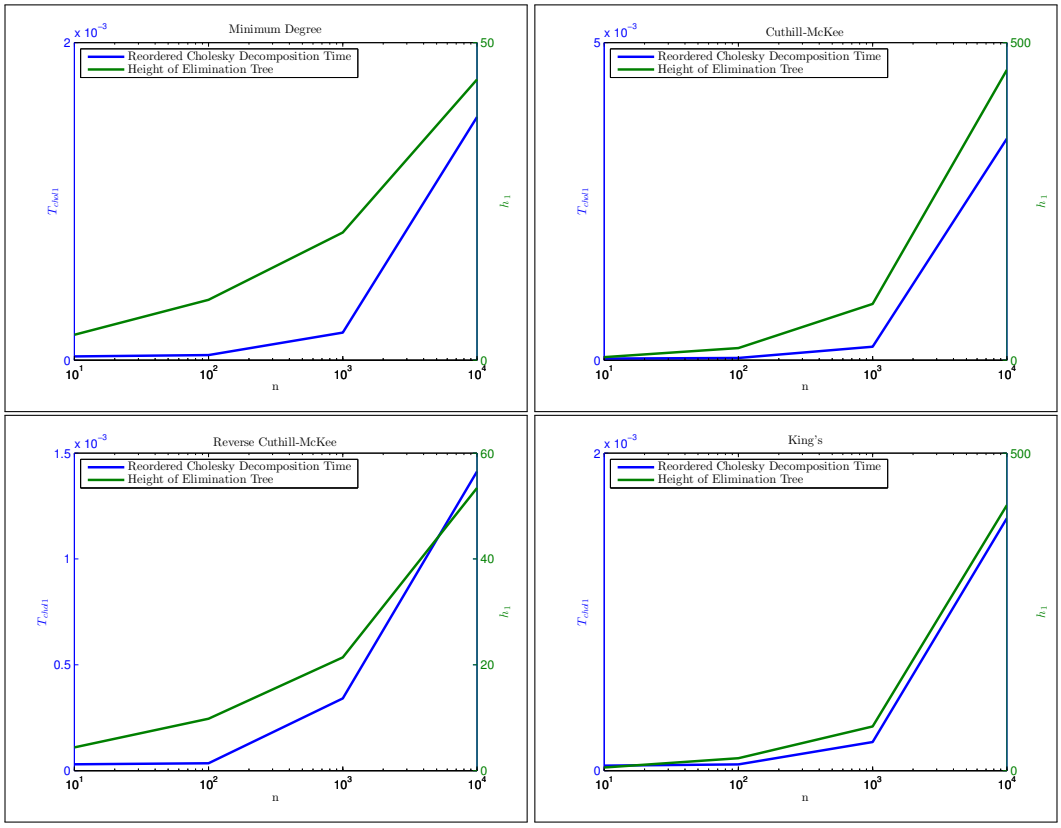

Figure 9: Height of elimination tree and Cholesky decomposition time.

5.1.2 Comparison Across Special Matrices

A comprehensive summary table of the results for special matrices is shown in the Appendix. Only the times (the first three columns) were averaged over ten iterations of each matrix.

A visual representation of the reorderings for the Mesh2EM5 is shown below. Note that the reordering algorithms produce a more complex structure, as compared to the simple collapse of the elements to the diagonal in the case of random matrices.

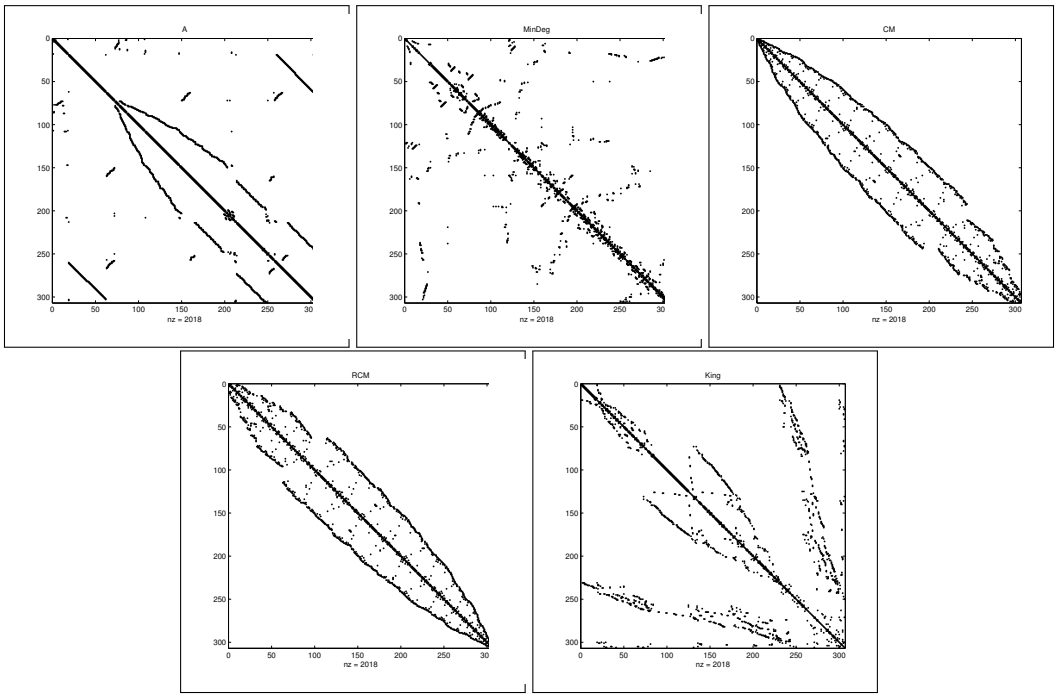

Figure 10: Mesh2EM5, and applications of the reordering algorithms.

Matrix Reordering Time

In studying the time for reordering each of the three special matrices, we find that the Minimum Degree algorithm is significantly slower than the other algorithms, and in fact is an order of 10^3 slower for matrices with side-length $\approx c \cdot 10^3$ as compared to the smallest matrix (with side-length $\approx c \cdot 10^2$) for scalar c. This is likely because the large matrices have long adjacency sets per eliminated node. Thus the Minimum Degree algorithm spends more time updating the degrees of the snodes in the adjacency sets. Furthermore, the execution time of the Minimum Degree algorithm is also significantly lower for random matrices of similar size.

The time of reordering for each of the special matrices, for each algorithm, can be seen in the figure below. Note that the y-axis is on a $\log$-scale, so times that are originally between 0 and 1 second appear negative.

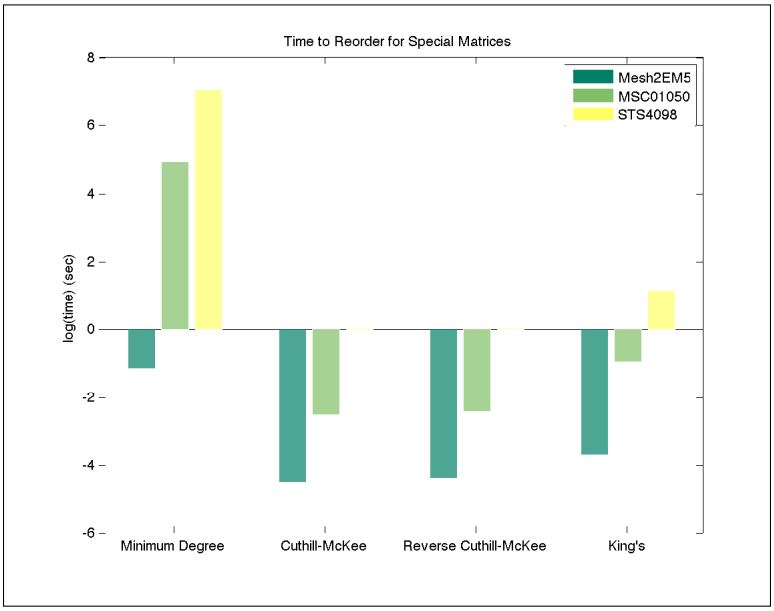

Figure 11: Time for reordering special matrices.

Cholesky Decomposition Runtime

T_{chol_1} exhibits interesting trends: the Cholesky decomposition runtimes for reordered matrices are generally faster. However, for the MSC01050, the Cholesky decomposition runtime for the original matrix is faster, for all algorithms. Our tests did not show any consistent superior performance across matrices for any particular algorithm.

Bandwidth and Profile

We find that the change in bandwidth for the CM and RCM algorithms are the same. This makes sense since the key difference comes from reversing the permutation vector in the final step of the RCM. As a result, the bandwidth - a sort of summary measurement of the reordered matrix - will remain the same, regardless of the type of matrix they are operated on. On the other hand, the change in profile for RCM is significantly higher than for CM in all matrices. This is consistent with our expectations, based on the design of the RCM algorithm, as discussed in 3.2.

Additionally, we see that the change in profile for King's algorithm is larger than all other reordering algorithms for two of the three matrix types, and specifically for the largest matrices. This is consistent with the theory behind King's algorithm: in it, a deliberate decision is made to trade low bandwidth in order to get low profile measures. The benefit of doing this is that nodes within a cluster are reordered "together" (as opposed to, for example, RCM). This was discussed in more detail in 3.3.

The figure below validates our intuition and the algorithmic structure of Minimum Degree, CM, RCM and King's. The $\triangle bandwidth$ and $\triangle profile$ metrics for each of the special matrices, for each algorithm, can be seen in the figure here:

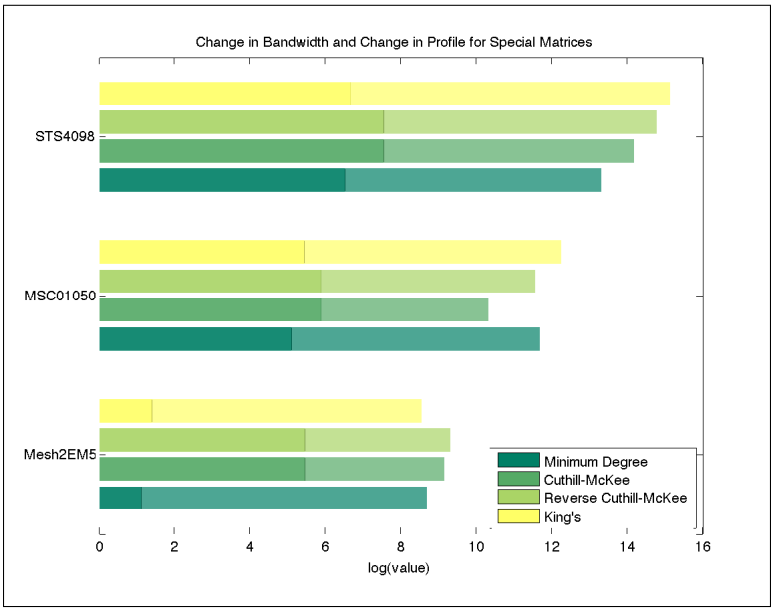

Figure 12: $\triangle profile$ and $\triangle bandwidth$ for special matrices.

5.2 Impact of Reordering on Computation

The results of measuring the impact of reordering on computation are shown in the figure below. We measure the average error for 5 iterations of random sparse n by n matrices for size $n = 10 : 10 : 2000$. There does not appear to be a meaningful difference in the performance of any of the permuted matrices as compared to the original. The small variations in the accuracy occur at distinctive levels, which indicate that the differences are merely due to error at machine precision levels caused by truncation.

This leads us to conclude that in the case of sparse random matrices, which generally exhibit very low levels of connectivity, the reordering schemes do not offer any significant advantage to the computational accuracy of operations using the Cholesky decomposition.

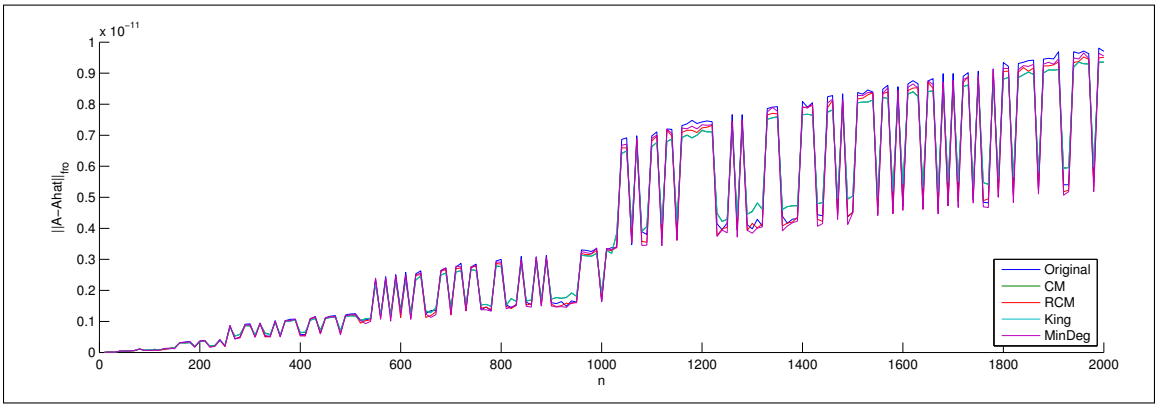

Figure 13: Error resulting from reconstructing the reordered matrix.

We also test this error metric on the special matrices, which have a much higher level of structural connectivity. These results reveal the potential impact a choice of reordering algorithm can have on accuracy. The figure below shows the results for each of the three special test matrices. Though the overall error levels are reasonably small, there does appear to be a more distinct difference between the results of some of the ordering algorithms compared to the original matrix. In most cases, the error for any of the reordered matrices is somewhat lower than the original, but no single method stands out as consistently producing the least error. This is likely due to structural differences in each matrix, such as the level of connectivity, which will depend on the real-world application of the matrix.

This measure is therefore yet another consideration in choosing a reordering algorithm, and some level of initial testing is recommended to determine the most numerically accurate reordering method for a problem-specific matrix.

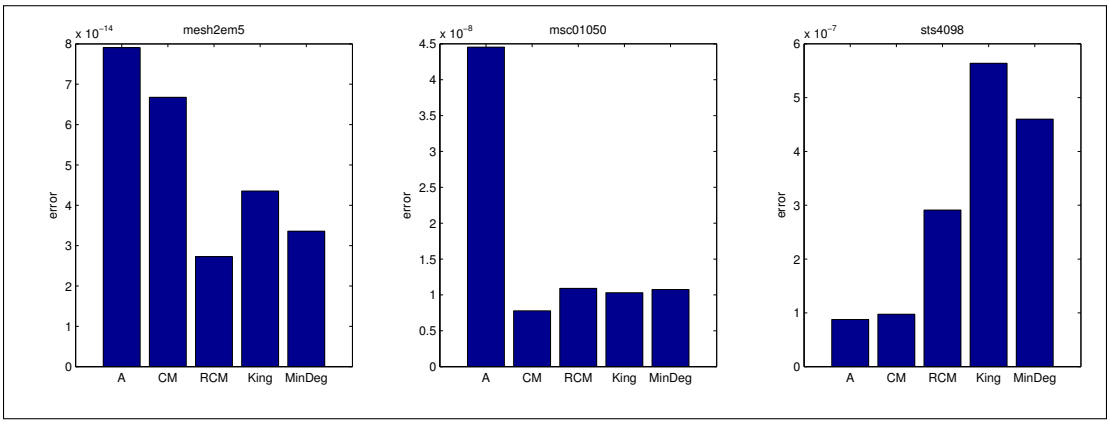

Figure 14: Error resulting from reconstructing the special reordered matrix.

6 Conclusion

Our implementations of the reordering algorithms described in this paper exhibit the expected qualities based on the existing literature. Through our many tests across six metrics and varying-sized random and structured matrices, we found that the RCM performed best, *on average*. However, there is no single, optimal algorithm that outperforms the others across all matrices, for all cost metrics. As demonstrated by our exploration of the special matrices, ultimately, the choice of heuristic is highly-dependent on the application of the matrix under consideration.

References

[1] Cannizzo, F. STS4098. UF Sparse Matrix Collection. MAT

[2] Cuthill, E., and J. McKee. "Reducing the Bandwidth of Sparse Symmetric Matrices." Proceedings of the 1969 24th National Conference ACM '69 (1969): 157-72. ACM Digital Library. Web.

[3] Fang, Haw-ren, and Dianne P. O'Leary. "Modified Cholesky Algorithms: A Catalog with New Approaches." University of Maryland Technical Report CSTR-4807 (2006): n. pag. Web.

[4] George, Alan, and Joseph W. H. Liu. Computer Solution of Large Sparse Positive Definite Systems. Englewood Cliffs, NJ: Prentice-Hall, 1981. Print.

[5] Gibbs, Norman E., William G. Poole, Jr., and Paul K. Stockmeyer. "An Algorithm for Reducing the Bandwidth and Profile of a Sparse Matrix." SIAM Journal on Numerical Analysis 13.2 (1976): 236. Web.

[6] Gupta, Anshal, George Karypis, and Vipin Kumar. "Highly Scalable Parallel Algorithms for Sparse Matrix Factorization." IEEE Transactions on Parallel and Distributed Systems 8.5 (1997): 502-20. Web.

[7] Grimes, R. MSC00726. UF Sparse Matrix Collection. MAT.

[8] Heggernes, P., S.C. Eisenstat, G. Kumfert, and A. Pothen. The Computational Complexity of the Minimum Degree Algorithm. Rep. 42nd ed. Vol. 2001. Hampton: Langley Research Center, VA. Web.

[9] Ingram, Stephen. Minimum Degree Reordering Algorithm: A Tutorial. N.p., n.d. Web. 15 Nov. 2013.

[10] "Introduction to Fill-In." Fill In. N.p., n.d. Web. 12 Nov. 2013.

[11] Kaveh, A., and P. Sharafi. "A Simple Ant Algorithm for Profile Optimization of Sparse Matrices." Asian Journal of Civil Engineering (Building and Housing) 9.1 (2007): 35-46. Web.

[12] King, Ian P. "An Automatic Reordering Scheme for Simultaneous Equations Derived from Network Systems." International Journal for Numerical Methods in Engineering 2.4 (1970): 523-33. Print.

[13] Lin, Wen-Yang, and Chuen-Liang Chen. "N Optimal Fill-Preserving Orderings of Sparse Matrices for Parallel Cholesky Factorizations." IEEE (2000): n. pag. Web.

[14] Markowitz, H. M. "The Elimination Form of the Inverse and Its Application to Linear Programming." Management Science 3.3 (1957): 255-69. Web.

[15] Mueller, Christopher, Benjamin Martin, and Andrew Lumsdaine. "A Comparison of Vertex Ordering Algorithms for Large Graph Visualization." APVIS (2007): n. pag. Web.

[16] Pissanetzky, Sergio. Sparse Matrix Technology. London: Academic, 1984. 96-97. Print.

[17] Pothen. MESH2EM1. UF Sparse Matrix Collection. MAT.

[18] "Sparse Matrices Demo." Mathworks.com. MATLAB, n.d. Web. 14 Dec. 2013. [http://www.mathworks.com/products/matlab/examples.html?file=/products/demos/shipping/matlab/sparsity.html].

[19] Zundel, Detlev. "Implementation and Comparison of Three Bandwidth Optimizing Algorithms on Distributed Memory Parallel Computer." Interner Berich 75.01 (2000): n. pag. Web.

Appendix

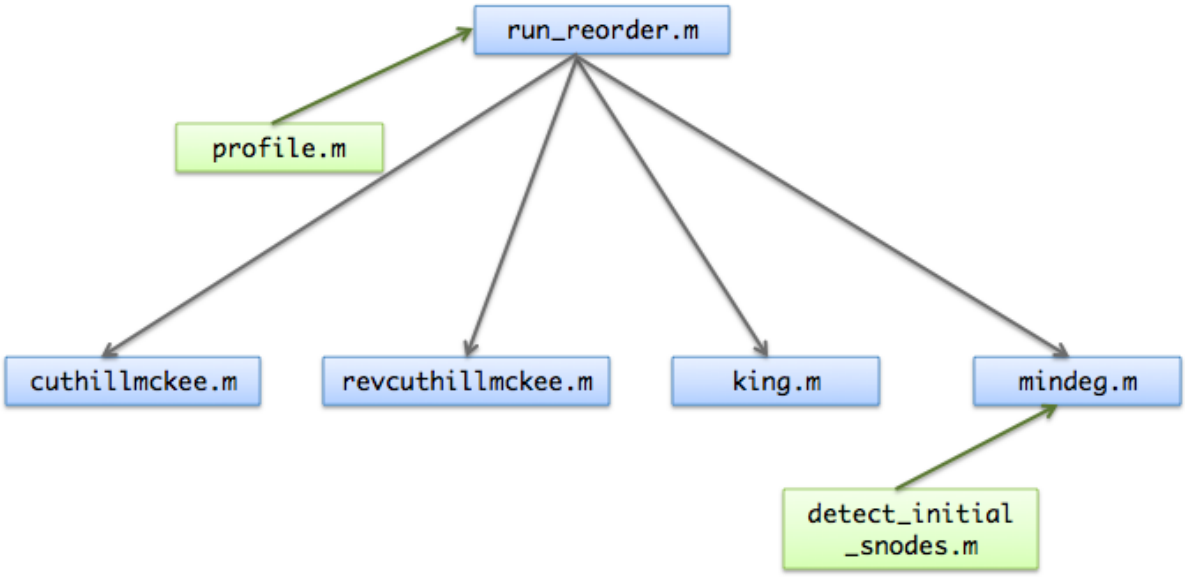

Figure 15: Visual representation of code; blue=functions executed; green=sub-functions .

Table 1: Cost Metric Symbols and their Meanings

Symbol	Meaning
T	Average time to reorder (sec)
T_{chol_0}	Average time to run Cholesky decomposition, original (sec)
T_{chol_1}	Average time to run Cholesky decomposition, reordered (sec)
θ_0	Average fill-in between Cholesky and original
θ_1	Average fill-in between Cholesky and reordered
$\triangle\theta$	Average change in fill-in
h_0	Average height of elimination tree
h_1	Average height of elimination tree, reordered
$\triangle h$	Average change in height of elimination tree
B_0	Average bandwidth
B_1	Average bandwidth, reordered
$\triangle B$	Average change in bandwidth
ϕ_0	Average profile
ϕ_1	Average profile, reordered
$\triangle\phi$	Average change in profile

Table 2: Summary of Results (each entry corresponds to an average over 10 matrices)

Algorithm	**Cost Metric**	$N = 1$	$N = 2$	$N = 3$	$N = 4$
Minimum Degree	T	0.008865817	0.02916505	0.563290863	37.22235924
	T_{chol_0}	3.69×10^{-5}	5.06×10^{-5}	2.76×10^{-4}	0.00271637
	T_{chol_1}	2.38×10^{-5}	3.23×10^{-5}	1.74×10^{-4}	0.001531277
	θ_0	1.1	35.5	4.52×10^2	4.36×10^3
	θ_1	0	1.6	1.4	14.5
	$\triangle\theta$	1.1	33.9	4.50×10^2	4.34×10^3
	h_0	3.8	10.9	22.1	45.7
	h_1	4	9.5	20.1	44.2
	$\triangle h$	0.2	1.4	2	1.5
	B_0	6.9	89.4	9.70×10^2	9.91×10^3
	B_1	4	47.8	5.83×10^2	6.16×10^3
	$\triangle B$	2.9	41.6	3.87×10^2	3.75×10^3
	ϕ_0	14.6	1.32×10^3	1.33×10^5	1.32×10^7
	ϕ_1	7.6	317	3.25×10^4	3.24×10^6
	$\triangle\phi$	7	1.01×10^3	1.00×10^5	9.99×10^6
Cuthill-McKee	T	0.001018471	0.002698167	0.057423904	5.445143345
	T_{chol_0}	4.01×10^{-5}	5.20×10^{-5}	2.70×10^{-4}	0.003466301
	T_{chol_1}	2.56×10^{-5}	3.52×10^{-5}	2.12×10^{-4}	0.003487462
	θ_0	1.8	32.1	4.18×10^2	4.66×10^3
	θ_1	0.8	39.9	692	1.30×10^4
	$\triangle\theta$	1	7.8	2.74×10^2	8.32×10^3
	h_0	3.7	8.9	20.7	51.9
	h_1	4.8	19.1	88.5	4.57×10^2
	$\triangle h$	1.1	10.2	67.8	4.05×10^2
	B_0	7.5	87.5	9.64×10^2	9.88×10^3
	B_1	2.6	5.6	12.2	30.8
	$\triangle B$	4.9	81.9	9.52×10^2	9.84×10^3
	ϕ_0	15.2	1.31×10^3	1.33×10^5	1.32×10^7
	ϕ_1	5.7	90	1.19×10^3	1.80×10^4
	$\triangle\phi$	9.5	1.22×10^3	131879	1.32×10^7
Reverse CM	T	0.001649149	0.002963737	0.057776124	4.380610848
	T_{chol_0}	5.67×10^{-5}	1.00×10^{-4}	5.11×10^{-4}	0.002546748
	T_{chol_1}	3.01×10^{-5}	3.50×10^{-5}	3.41×10^{-4}	0.001413201
	θ_0	1.7	32.9	4.45×10^2	4.33×10^3
	θ_1	0.1	0.3	2.4	18.6
	$\triangle\theta$	1.6	32.6	4.42×10^2	4.32×10^3
	h_0	3.5	9.4	21.8	42.7
	h_1	4.4	9.8	21.4	53.4
	$\triangle h$	0.9	0.4	0.4	10.7
	B_0	7.3	92.1	9.63×10^2	9.88×10^3
	B_1	2.6	4.8	11.5	24.4
	$\triangle B$	4.7	87.3	9.52×10^2	9.85×10^3
	ϕ_0	15.9	1.44×10^3	1.33×10^5	1.33×10^7
	ϕ_1	5.6	63.1	8.66×10^2	1.11×10^4
	$\triangle\phi$	10.3	1.38×10^3	132186	1.32×10^7

Table 3: Summary of Results Continued from Table 2

Algorithm	**Cost Metric**	$N = 1$	$N = 2$	$N = 3$	$N = 4$
King	T	9.24×10^{-4}	0.002885633	0.057957624	4.38259303
	T_{chol_0}	4.58×10^{-5}	5.07×10^{-5}	3.32×10^{-4}	0.00272671
	T_{chol_1}	3.12×10^{-5}	3.90×10^{-5}	1.80×10^{-4}	0.001589053
	θ_0	2.7	30.8	367	4767
	θ_1	0.5	21.1	2.90×10^2	4.55×10^3
	θ_1	2.2	9.7	77.3	2.19×10^2
	$\triangle\theta$	4.4	9.3	17	54.1
	h_0	5	19.5	69.6	4.18×10^2
	h_1	0.6	10.2	52.6	3.64×10^2
	$\triangle h$	7.6	86.9	9.65×10^2	9.85×10^3
	B_0	2.8	7.2	31	2.11×10^2
	B_1	4.8	79.7	9.34×10^2	9.64×10^3
	ϕ_0	18.4	1.40×10^3	1.32×10^5	1.32×10^7
	ϕ_1	6.2	71.7	7.90×10^2	9.55×10^3
	$\triangle\phi$	12.2	1.33×10^3	1.31×10^5	1.32×10^7

Table 4: Summary of Results for Special Matrices

Algorithm	**Matrix**	T	T_{chol_0}	T_{chol_1}	θ_0	θ_1	$\triangle\theta$
Minimum Degree	mesh2em5	0.3204	0.026436495	9.64×10^{-4}	10939	5416	5523
	msc01050	1.36×10^2	0.002424151	0.008687763	16993	30430	13437
	sts4098	1.1384×10^3	0.790709579	0.10796706	4644719	849311	3795408
Cuthill-McKee	mesh2em5	0.3204	0.037868208	0.001366437	10939	10380	559
	msc01050	0.009660564	0.00274581	0.031392243	16993	295561	278568
	sts4098	0.950648615	0.766366565	0.490187747	4644719	3780425	864294
Reverse CM	mesh2em5	0.071315665	8.52×10^{-4}	6.63×10^{-4}	10939	8422	2517
	msc01050	0.010519947	0.002444281	0.012476228	16993	140354	123361
	sts4098	0.982006881	0.773638244	0.145435832	4644719	1123854	3520865
King's	mesh2em5	0.315154061	0.001183878	0.00116088	10939	14653	3714
	msc01050	0.016832284	0.002683467	0.007512954	16993	120959	103966
	sts4098	2.770592393	0.85372956	0.169812213	4644719	1483522	3161197

Table 5: Summary of Results for Special Matrices, Continued

Algorithm	**Matrix**	h_0	h_1	$\triangle h$	B_0	B_1	$\triangle B$	ϕ_0	ϕ_1	$\triangle\phi$
Minimum Degree	mesh2em5	250	270	20	286	283	3	20548	14698	5850
	msc01050	108	265	157	786	948	162	337923	220772	117151
	sts4098	3894	2008	1886	3324	3988	664	5217389	4625474	591915
Cuthill-McKee	mesh2em5	250	306	56	286	56	230	20548	11236	9312
	msc01050	108	1050	942	786	432	354	337923	308135	29788
	sts4098	3894	4098	204	3324	1464	1860	5217389	3814557	1402832
Reverse CM	mesh2em5	250	295	45	286	56	230	20548	9633	10915
	msc01050	108	720	612	786	432	354	337923	234754	103169
	sts4098	3894	2298	1596	3324	1464	1860	5217389	2650283	2567106
King's	mesh2em5	250	306	56	286	282	4	20548	15509	5039
	msc01050	108	1050	942	786	1012	226	337923	133538	204385
	sts4098	3894	4098	204	3324	4095	771	5217389	1520478	3696911