Improving Parallel Ordering of Sparse Matrices using Genetic Algorithms

April 23, 2005

Wen-Yang Lin

Department of Information Management

I-Shou University

Kaohsiung County, Taiwan 84008, ROC

Email: wylin@isu.edu.tw

Abstract

In the direct solution of sparse symmetric and positive definite linear systems, finding an ordering of the matrix to minimize the *height* *of the elimination tree* (an indication of the number of parallel elimination steps) is crucial for effectively computing the Cholesky factor in parallel. This problem is known to be *NP*-hard. Though many effective heuristics have been proposed, the problems of how good these heuristics are near optimal and how to further reduce the height of the elimination tree remain unanswered. This paper is an effort for this investigation. We introduce a genetic algorithm tailored to this parallel ordering problem, which is characterized by two novel genetic operators, adaptive merge crossover and tree rotate mutation. Experiments showed that our approach is cost effective in the number of generations evolved to reach a better solution in reducing the height of the elimination tree.

Keywords. Sparse matrix ordering, parallel factorization, genetic algorithms, elimination tree.

1 Introduction

In the direct solution of sparse symmetric and positive definite linear systems, it has been recognized that parallel pivoting (pivots that can be eliminated simultaneously) is crucial for effectively computing the Cholesky factor in parallel (Calahan 1973, Heath 1991). When the sparse matrix is viewed as a graph, the parallel pivoting problem is equivalent to finding an ordering of the vertices to minimize the *height of the elimination tree* (an indication of the number of parallel elimination steps). For general graphs this problem is known to be *NP*-hard. Researchers are devoted to finding effective heuristics and usually adopt a modular approach. First, a fill-reducing ordering, such as minimum degree (George 1989) or nested dissection (George 1973), is applied. Then, based on this ordering, an equivalent reordering (which preserves the fills) is produced such that the reordered matrix can be factored effectively in parallel (Lewis 1989, Liu 1989). The modular approach, though would generate an ordering that minimizes the number of parallel elimination steps and preserve the fill, has prevented further exploitation of better solutions because of the fill preserving constraint. Even if the fill preserving property is neglected, the problem of how to further reduce the height of the elimination tree remains unanswered.

In recent years, a large body of work exists in applying genetic algorithms to different application areas (Goldberg 1989, Mitchell 1996). In contrast, not much work has been done on the parallel ordering problem. In a previous work, we have found the possibility of applying genetic algorithms to this problem. This paper is a further investigation in this problem.

We propose a genetic algorithm that is tailored to the parallel ordering problem and is characterized by two novel genetic operators, *adaptive merge crossover* and *tree rotate mutation*. Experiments showed that our approach is cost effective in the number of generations evolved to reach a better solution that has considerable improvement in reducing the height of the elimination tree.

An outline of the paper is as follows. First, we describe the background for parallel sparse Cholesky factorization and the parallel ordering problem. In Section 3, we describe the proposed genetic algorithm and discuss some implementation issues. In Section 4 we present the experimental results on a set of test matrices from the Harwell-Boeing collection. Finally, Section 5 states the conclusions.

2 Background

Consider a system of linear equations

$$Ax = b,$$

where A is an $n \times n$ sparse symmetric and positive definite matrix, b is a given vector, and x is the unknown vector to be solved. It is known that many scientific applications give rise to such a system of linear equations. In the direct solution of the linear systems, A is usually first decomposed into LL^T, which is known as Cholesky factorization, where L is the lower triangular. Then the solution vector x is computed by solving two triangular systems $Ly = b$ and $L^T x = y$ (George 1981). Since A is sparse, meaning that most of its entries are zeros and, as a variable is eliminated during factoring A into LL^T, some entries that are initially zero in A may become nonzero in L. These entries are known as *fill* or *fill-in*.

As an example, consider the following system

$$\begin{bmatrix} 4 & 2 & 2 & 0.5 & 2 \\ 2 & 2 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 0.5 & 0 & 0 & 0.875 & 0 \\ 2 & 0 & 0 & 0 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \\ 6 \\ 4 \\ 2 \end{bmatrix}.$$

The Cholesky factor of the coefficient matrix is

$$\begin{bmatrix} 2 & & & & \\ 1 & 1 & & & \\ 1 & \mathbf{-1} & 1 & & \\ 0.25 & \mathbf{-0.25} & \mathbf{-0.5} & 0.5 & \\ 1 & \mathbf{-1} & \mathbf{-2} & \mathbf{-3} & 1 \end{bmatrix},$$

where the bold faced values denote the fill.

For efficient use of computer storage and processing time, it is desirable for the amount of fill to be small, and only the nonzeros in A and L are stored and operated.

Calahan (Calahan 1973) was the first one to recognize that in parallel computation of the Cholesky factor of a sparse matrix, pivots of no data dependency can be eliminated simultaneously. And the more pivots being processed simultaneously, the less computation time is spent on the factorization. Research has shown that the order of equations has great impact on exploiting the parallel pivoting (Jess 1981). The problem of finding a good ordering of the sparse set of equations appropriate for parallel factorization is called the *parallel pivoting* or *parallel ordering* problem. For example, consider the tridiagonal system shown in Figure 1(a). Let us call the ordering that preserves the tridiagonal the *natural ordering.* Under the natural ordering, we observe that each column except the first one depends on the immediately preceding column and must be computed sequentially during the factorization. The ordering exhibits no parallelism. Consider the same matrix ordered by a nested dissection ordering as shown in Figure 1(b). Now columns 1, 2, 4 and 5 have no preceding dependence and thus can be computed simultaneously.

By the combinatorial nature of the sparse ordering problem, it is convenient to view a sparse matrix as a graph. Let A be a $n \times n$ sparse symmetric and positive definite matrix. Define its associated graph, $G(A) = (V, E)$, as an undirected graph with the vertex set $V = \{1, 2, \ldots, n\}$ and the edge set $E = \{(i, j) \mid \text{if } a_{ij} \neq 0\}$. An ordering α of V is a bijection $\alpha : \{1, 2, \ldots, n\} \rightarrow V$. Figure 2 shows an 8×8 sparse matrix and its associated graph ordered by the natural ordering.

Given a graph ordered by α, $G_\alpha(A) = (V, E, \alpha)$, we call the adjacency graph of the resulting filled matrix $F = L + L^T$ the *filled graph* of $G_\alpha(A)$, where L denotes the Cholesky factor. Figure 3 shows the Cholesky factor of the matrix in Figure 2 ordered by 3 2 6 8 1 5 7 4 and the filled graph, where symbol '$\times$' denotes a nonzero and '$\bullet$' a fill.

To exploit the inherent parallelism among pivots in sparse matrix factorization, a commonly used structure is the *elimination tree* (Jess 1981, Schreiber 1982). Given a graph ordered by α, $G_\alpha(A) = (V, E, \alpha)$, the elimination tree $T_\alpha(A)$ is a tree containing the same nodes as the filled graph of the Cholesky factor of A and, for each node k with $k < n$, its parent node is p, where $p = \min\{j \mid j > k \text{ and } l_{jk} \neq 0\}$. That is, the parent node of k is the first nonzero off the diagonal on column k. The effect of reordering for reducing elimination tree height is illustrated in Figure 4, where the matrix ordered by 3 2 6 8 1 5 7 4 has decreased the height by one.

The elimination tree structure exhibits in a minimal form of the dependencies among pivots in a sparse matrix. The height of the elimination tree indicates essentially the minimum number of parallel elimination steps to complete the factorization. From this viewpoint, the parallel ordering problem can be regarded as finding an ordering of the associated graph of a sparse matrix to minimize the height of the elimination tree.

For a general graph, the ordering problem of minimizing elimination tree height is known to be *NP*-hard. In the literature there are many effective heuristic methods. But how well these methods are near to the optimal remains unanswered. Our intention in this work has two aspects: devise a genetic algorithm to improve the quality of an ordering generated by some heuristic method and to appraise the effectiveness of the heuristic method.

3 The Proposed Genetic Algorithm

The genetic algorithm (Holland 1992) is an innovative approach composed of many biological imitations, such as the chromosome representation, genetic operators, population selection, and fitness functions. Our work is based on a modified simple genetic algorithm (Goldberg 1989), which is described in Algorithm 1. Below, we state the primary issues involved in applying the GA to the parallel ordering problem, and propose new genetic operators tailored to this problem.

ALGORITHM 1. *A GA for parallel pivoting*

INPUT: A heuristic ordering and an adjacent structure of the sparse matrix

OUTPUT: A resulting ordering

```
begin
    Initialize the parameters;
    Generate a population $P$ randomly;
    $generation \leftarrow 1$;
    while $generation \leq max\_gen$ do
        Clear the new population $P'$;
        Use a fitness function $\mathcal{F}(\cdot)$ to evaluate each individual in $P$;
        while $|P'| \leq population\_size$ do
            Select two parents from $P$;
            Perform crossover;
            Perform mutation;
            Place the offspring into $P'$;
        endwhile
        $P \leftarrow P'$; /* Replace the old population. */
        $generation \leftarrow generation + 1$;
    endwhile
```

end

3.1 Chromosome Representation

The encoding scheme is the first step in, and a key part of using GAs. To facilitate the crossover and mutation operations, and to enhance the performance of the algorithm, a chromosome representation that stores current solution states is desirable. Choosing an appropriate chromosome representation is not trivial. It depends on the problem of concern and the size of the search space. In this work, we use a straightforward representation: each ordering is represented as a chromosome. That is, a chromosome $\chi = x_1x_2 \ldots x_n$ corresponds to an ordering of $G(A)$. For example, consider the matrix in Figure 3, which is ordered by 3 2 6 8 1 5 7 4. The corresponding chromosome is $\chi = 32681574$.

3.2 Fitness Evaluation

The fitness function is important for devising an effective GA. Indeed, it may be the most time-consuming and critical component in GAs. The fitness function evaluates each chromosome and generates values representing the degree of fitness. Chromosomes with smaller fitness values are usually discarded to increase the population superiority. A simple evaluation for the parallel ordering problem is using the height of the elimination tree. That is, given a chromosome χ, we define the fitness function $\mathcal{F}$ below

$$\mathcal{F}(\chi) = height(T_{\chi}(A)).$$

Algorithm 2 describes the function for computing the elimination tree height, where the tree structure is accomplished in the form of a $Parnt(*)$ vector. Vector $Heght(*)$ stores the height of the subtree rooted at each node in the elimination tree. For example, the $Parnt$ and $Heght$ vectors for the elimination tree in Figure 4 with respect to ordering 3 2 6 8 1 5 7 4 are described below

i	1	2	3	4	5	6	7	8
$Parnt$	5	1	4	0	7	7	4	5
$Heght$	2	1	1	5	3	1	4	1

where the node i with $Parnt(i) = 0$ denotes the root.

ALGORITHM 2. *Elimination Tree Height*

INPUT: An ordering χ and the adjacency structure of the sparse matrix

OUTPUT: The elimination tree height

```
begin
    for $i = 1$ to $n$ do
        $ni \leftarrow \chi(i)$;
        $Parnt(ni) \leftarrow 0$;
        $Heght(ni) \leftarrow 1$;
        /* Add the $i$-th node $x_i$ into the elimination tree. */
        **for** $k \in Adj_{G(A)}(ni)$ and $\chi(k)^{-1} < i$ **do**
            $r \leftarrow k$;
            /* Trace in the partial elimination tree constructed so far to find
                the root of the subtree containg the node k. */
            **while** $Parnt(r) \neq 0$ and $Parnt(r) \neq ni$ **do**
                $r \leftarrow Parnt(r)$;
            **if** $Parnt(r) = 0$ **then**
                /* Add $x_i$ to the elimination tree by setting $x_i$ as the root of the
                    found subtree. Also update the height of the subtree. */
                $Parnt(r) \leftarrow ni$;
                **if** $Heght(ni) < Heght(r) + 1$ **then**
                    $Heght(ni) \leftarrow Heght(r) + 1$;
            **endif**
        **endfor**
    **endfor**
    **return**($Heght(ni)$);
**end**
```

The algorithm starts with a null elimination tree. At each step i, the ith node, i.e., $\chi(i)$, is added to the elimination tree in the manner of a bottom-up traverse along the current $Parnt$ structure to identify all its current children. The $Parnt$ and the $Heght$ vectors are updated accordingly to reflect this partial elimination tree. This procedure is repeated till the whole elimination tree is constructed. Figure 5 shows the stages in executing Algorithm 2 on the example in Figure 2, where $\chi = 32681574$.

The most time-consuming part of Algorithm 2 is the while loop for tracing the children of the ith node. A technique called *path compression* can be used to speed up the traverse process. With this modification, the complexity of Algorithm 2 can be reduced to $O(|E(A)| \log_2 n)$. We adopt this technique in our implementation. The interested reader should refer to (Liu 1986) for the details.

3.3 Crossover

The crossover operation is perhaps the most distinguishing feature of GAs. By exchanging genes in a pair of individuals (parents) chosen from the population, the crossover mimics the ecological mating process to generate offspring. In the literature, there are diverse forms of crossover and they usually are tailored to the chromosome representations (Mitchell 1996). One class, called the sequencing crossover, is dedicated to order-based GAs, in which all solutions are encoded as a permutation of a list. Typical sequencing crossovers include *edge recombination* (ER) or *enhanced edge recombination* (EER), *order crossover* #1 (OX1), *order crossover* #2 (OX2), *partially mapped crossover* (PMX), *cycle crossover* (CX), and *position based crossover* (POX). The main difference between these operators is the information to be preserved during recombination. The edge recombination operator is designed to preserve the adjacency information while the others are more sensitive to the relative order (OX1, OX2, and POX) or absolute position (PMX and CX). A comprehensive study of these operators was conducted in (Starkweather 1993).

Most sequencing crossovers are based on the local precedence among the genes in a chromosome and dependent only on the contents of the chromosomes. Recall that our purpose is, given an ordering generated by some heuristic method such as minimum degree or a hybrid of minimum degree and minimum height, to improve the ordering quality, i.e., reduce the height of the elimination tree. This heuristic solution can be seen as a local minimum in the search space. One way to make this local minimum conducive to the optimal solution is incorporating the local minimum in the initial population and applying one of the sequencing crossovers in subsequent evolutionary processes. In a preliminary work, we have considered this approach and found that it usually needs much more recombinations to reach another better minimum. This means that the information concealed in the heuristic solution does not contribute effectively to the genetic search. Another way is generating all of the individuals in the initial population with the heuristic method. This approach seems to be promising but has two deficiencies. First, we have to apply the heuristic method many times with different seeds. This could be too time consuming. Second, the solutions obtained by the heuristic method usually have similar characteristics even with different seeds. The population diversity would be limited to a local landscape in the search space and may not explore further areas.

To alleviate these problems, we adapted Blanton Jr. and Wainwright's merge operator (Blanton Jr. 1993) in the present work, which was originally proposed to tackle the optimization of vehicle routing problems with constraints. Given a solution encoded as an n-gene chromosome, a global precedence relationship among the genes is first created according to the constraint. The precedence vector is used as a guide for generating offspring. For example, consider the following two chromosomes, $p1$ and $p2$, and a precedence vector ω.

$$\begin{array}{rcccccccc}
p1: & \underline{4} & 3 & 2 & 6 & 1 & 7 & 5 & 8 \\
p2: & \underline{5} & 1 & 7 & 8 & 2 & 3 & 4 & 6 \\
\omega: & 3 & 2 & 7 & \underline{4} & 1 & 8 & 6 & \underline{5} \\
c: & 4 & & & & & & &
\end{array}$$

The first gene in $p1$, 4, has earlier precedence in ω than the first gene of $p2$, 5. Therefore the offspring inherits its first gene from $p1$. The two genes, 4 and 5, in $p2$ are then swapped to maintain the validity. The process continues until the offspring is filled with genes. The final result is shown below.

$$c: \quad 4 \quad 3 \quad 2 \quad 8 \quad 7 \quad 1 \quad 5 \quad 6$$

For the purpose of our work, the precedence vector is initially set to the solution obtained by some heuristic method and replaced by the best chromosome of the population in each subsequent generation. This effects the genetic search always being directed by the current best solution. As one might expect, the search space would converge more quickly to an optimal. We called this modified operator the *adaptive merge crossover*, whose function is described in Algorithm 3.

ALGORITHM 3. *Adaptive merge crossover* (AMX)

INPUT: Two randomly selected chromosomes $p1$, $p2$ and the best chromo-

```
some of the current population, $\chi$
OUTPUT: An offspring $c$
begin
    $\omega \leftarrow \chi$; /* $\omega$ is set to the current best chromosome. */
    for $i = 1$ to $n$ do
        /* Compare in $\omega$ the precedence of the $i$-th gene of $p1$ and $p2$. */
        if $\omega^{-1}(p1(i)) < \omega^{-1}(p2(i))$ then
            $c(i) \leftarrow p1(i)$; /* Inherit the $i$-th gene from $p1$. */
            swap the two genes, $p1(i)$ and $p2(i)$, in $p2$;
        else if $\omega^{-1}(p1(i)) > \omega^{-1}(p2(i))$ then
            $c(i) \leftarrow p2(i)$; /* Inherit the $i$-th gene from $p2$. */
            swap the two genes, $p1(i)$ and $p2(i)$, in $p1$;
        else
            $c(i) \leftarrow p1(i)$;
        endif
    return($c$);
end
```

3.4 Mutation

The mutation operator is used to randomly change some elements in a selected individual so as to yield additional genetic diversity to help the search process escape from local optimal traps. For order-based GAs, it is natural to imitate the mutation by exchanging two randomly chosen elements, called the Swapping Mutation (SWM), or by inverting a subsequence, called the Inversion Mutation (IVM). In our preliminary experiments, we found that these two mutations are not amenable to our adaptive merge crossover. The diversity effect is faint. Hence, we devise another method, called the *tree rotate mutation*.

As usual, a chromosome for mutation together with a mutation point is randomly selected. Let the selected chromosome be $x_1x_2 \ldots x_n$ and the mutation point be m. The tree rotate mutation is accomplished by moving x_m and those genes adjacent to x_m in the graph $G(A)$ with later precedence than x_m to form the latter part of this chromosome. Then the remaining genes form the former part, retaining their relative orders in the chromosome. The effect of this operation, as will be illustrated later, looks like rotating the elimination tree at the node adjacent to x_m and ordered last.

For example, consider the graph in Figure 2 and the ordering 3 2 6 8 1 5 7 4 in Figure 3. Assume that the mutation point is 5; therein the element is 1. The set of genes adjacent to 1 with later precedence is $\{5\}$. Thus the tree rotate mutation moves 1 and 5 to the latter part of the chromosome and forms the former part with other genes, both parts of genes retaining their relative orders. The resulting chromosome is 3 2 6 8 7 4 1 5. As Figure 6 shows, the elimination tree looks like it is undergoing a rotation at node 5, resulting in a decrement on the tree height. The mutation operator is described in Algorithm 4.

ALGORITHM 4. *Tree Rotate Mutation* (TRM)

INPUT: A randomly selected chromosome χ and a mutation point m

OUTPUT: The mutated chromosome χ

```
begin
    $j \leftarrow 1$; $k \leftarrow m$;
    for $i = m$ to $n$ do
        /* Check if $x_i$ is adjacent to $x_m$ with later precedence. */
        if $\chi(i) \in Adj_{G(A)}(\chi(m)) \cup \{\chi(m)\}$ then
            /* Keep $x_i$ to a temporary vector $\tau$. */
            $\tau(j) \leftarrow \chi(i)$;
            $j \leftarrow j + 1$;
        else
```

```
            /* Move $x_i$ to the later part of the chromosome. */
            $\chi(k) \leftarrow \chi(i)$;
            $k \leftarrow k + 1$;
        **endif**
    /* Copy those genes in vector $\tau$ to the former part. */
    **for** $i = 1$ **to** $j - 1$ **do**
        $\chi(i + k - 1) \leftarrow \tau(i)$;
    **return**($\chi$)
**end**
```

4 Experimental Results

The evaluation is against the well-known modular approach, Liu's multiple minimum degree (MMD) ordering (Liu 1985), followed by an implementation (Lewis 1989) of the Jess and Kees algorithm (Jess 1981) to minimize the elimination tree height. Given a sparse matrix, this approach finds a good fill-reducing ordering first, and then generates an equivalent reordering in minimizing the elimination tree height while preserving the fill. We choose the power network matrices in the Harwell-Boeing collection (Duff 1992) as the benchmark. Table 1 summarizes the characteristics of the test matrices.

All programs were coded in the *C* language. The experiment was performed on a PC with Intel Pentium-III 600MHz CPU. All evaluations were the average over 20 runs and were performed under the following parameter settings: maximum generation is set to 50 with population size of 100; the crossover rate is set to 0.6 while mutation rate is set to 0.1.

In our preliminary work, we have tested the six sequencing crossovers, EER, OX1, OX2, PMX, CX, and POX, discussed in (Starkweather 1993), and found that the position-based crossover (POX) is superior to the others while edge recombination performs the worst. Our result is consistent with that observed by Starkweather (Starkweather 1993): Position or order preserving operators were superior to adjacency preserving operators for problems that are more sensitive to the order of the sequence, e.g., scheduling problems.

For comparison, we consider four versions in the evaluation: POX+SWM (swapping mutation), POX+TRM, AMX+SWM, and AMX+TRM. Figures 5 and 6 depict the results of the population diversity and the fitness of the best solution, respectively, for matrix BCSPWR09; similar phenomena were observed for the other matrices. For a population P composed of N chromosomes, each of length n, the diversity, measuring the degree to which a population is nonhomogeneous, is defined below

$$div(P) = \frac{2}{nN(N-1)} \sum_{\chi,\chi' \in P \wedge \chi \neq \chi'} \sum_{i=1}^{n} [x_i = x'_i] \, ,$$

where x_i and x'_i denote the ith gene in chromosome χ and χ', respectively, and $[x_i = x'_i]$ is 1 if $x_i = x'_i$; otherwise 0.

As one might expect from the conductive nature of the AMX operator, the population diversity converges quickly. Figure 7 clearly depicts this phenomenon. The diversity, when SWM is used, converges very quickly to zero after only 20 generations. But through the TRM operator, the population diversity is maintained sufficiently high above a minimum. This shows that the swapping mutation is not effective as a global exploring mechanism when it is incorporated with the AMX operator.

When cooperating with the POX operator, however, there is no significant difference between TRM and SWM, both retaining high diversity of the population. This means that POX performs better in exploring search dynamics than AMX. The result in Figure 8 asserts this inference. As the graph shows, AMX+TRM outperforms both in the convergence speed and the solution quality, but the curve is stable after 12 generations. The curve of POX+SWM, though it converges slowly, reveals further possibility in light of its downstairs shape to reach a better solution for more evolutions. We performed more evolutions in another experiment and found that POX+SWM needs more than about 100 generations to reach a better solution than AMX+TRM. The best results and the minimum generation the best appears for all matrices are reported in Table 2. On average, our proposed GA reduced by 15% the elimination tree height obtained by minimum degree and Jess and Kees algorithm after 8 evolutionary generations. This essentially asserts the feasibility of applying genetic algorithms as an effective postprocess in improving the ordering quality for parallel factorization.

Additionally, we also compared our proposed GA with another well-known class of fill-reducing methods, nested dissection, which is better at generating a balanced elimination tree. Here, we choose one of the leading packages, METIS (Karypis 1998). As the graph in Figure 9 shows, AMX+TRM again outperforms both in the convergence speed and the solution quality than the other combinations, for matrice BCSPWR09; similar phenomena were observed for the other matrices. The best results and the minimum generation the best appears for all matrices are summarized in Table 3. On average, our proposed GA reduced by 6% the elimination tree height after 5 evolutionary generations. This illustrates that METIS is better than MMD and that our approach can be used to appraise the quality of different heuristic methods.

5 Conclusions

We have presented a genetic algorithm to improve the quality of a given heuristic ordering in reducing the height of an elimination tree. The proposed algorithm is characterized by two novel operators, the adaptive merge crossover and tree rotate mutation. Through these two operators, our approach can make use of the information concealed in a heuristic solution to aid the exploration of better solutions.

Experimental results on some test matrices from the Harwell-Boeing collection showed that considerable improvements in the elimination tree height can be obtained within a very limited number of generations, even though the given heuristic solution is a fill-preserving, minimum height ordering. Our method can be used as an effective postprocess to any heuristic ordering methods.

The ordering of sparse matrices for reducing elimination tree height we consider in this work is just one of the ordering problems for sparse matrix computation. Other classes such as ordering for minimizing the fill, the completion time, storage, communication or any combinations of these criteria deserve further investigation.

Acknowledgments

This research was supported in part by the National Science Council of ROC under grant NSC87-2213-E-214-005.

References

[1] D. Calahan, Parallel solution of sparse simultaneous linear equations, in *Proc. 11th Annual Allerton Conference Circuit and System Theory* (1973) 729-735.

[2] I.S. Duff, R.G. Grimes and J.G. Lewis, User′s guide for the Harwell-Boeing sparse matrix collection, CERFACS, Toulouse Cedex, France, 1992.

[3] A. George, Nested dissection of a regular finite element mesh, *SIAM J. Numer. Anal.* 10 (1973) 345-363.

[4] A. George and J.W.-H. Liu, *Computer Solution of Large Sparse Positive Definite Systems* (Prentice-Hall, 1981).

[5] A. George and J.W.-H. Liu, The evolution of the minimum degree ordering algorithm, *SIAM Review* 31 (1989) 1-19.

[6] D. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, 1989).

[7] M.T. Heath, E. Ng and B.W. Peyton, Parallel algorithms for sparse linear systems, *SIAM Review* 33 (1991) 420-460.

[8] J. Holland, *Adaptation in Natural and Artificial Systems* (MIT Press, 1992).

[9] J.A.G. Jess and H.G.M. Kees, A data structure for parallel L/U decomposition, *IEEE Trans. Comput.* 31 (1982) 231-239.

[10] J.L. Blanton Jr. and R.L. Wainwright, Multiple vehicle routing with time and capacity constraints using genetic algorithms, in *Proc. Int'l Conference on Genetic Algorithms* (1993) 452-459.

[11] G. Karypis and V. Kumar, METIS: a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, Department of Computer Science, University of Minnesota, 1998. http://www-users.cs.umn.edu/ karypis/metis

[12] J.G. Lewis, B.W. Peyton and A. Pothen, A fast algorithm for reordering sparse matrices for parallel factorization, *SIAM J. Sci. Stat. Comput.* 10 (1989) 1146-1173.

[14] J.W.-H. Liu, Modification of the minimum degree algorithm by multiple elimination, *ACM Trans. Math. Software* 11 (1985) 141-153.

[15] J.W.-H. Liu, A compact row storage scheme for Cholesky factors using elimination trees, *ACM Trans. Math. Software* 12 (1986) 127-148.

[16] J.W.-H. Liu, Reordering sparse matrices for parallel elimination, *Parallel Comput.* 11 (1989) 73-91.

[17] M. Mitchell, *An Introduction to Genetic Algorithms* (MIT Press, 1996).

[19] R. Schreiber, A new implementation of sparse Gaussian elimination, *ACM Trans. Math. Software* 8 (1982) 256-276.

[20] T. Starkweather, et al, A comparison of genetic sequencing operators, in *Proc. Int'l Conference on Genetic Algorithms* (1993) 69-76.

$$\begin{pmatrix}
\times & \times & & & & & \\
\times & \times & \times & & & & \\
& \times & \times & \times & & & \\
& & \times & \times & \times & & \\
& & & \times & \times & \times & \\
& & & & \times & \times & \times \\
& & & & & \times & \times
\end{pmatrix}
\qquad
\begin{pmatrix}
\times & & \times & & & & \\
& \times & \times & & & & \times \\
\times & \times & \times & & & & \\
& & & \times & & \times & \\
& & & & \times & \times & \times \\
& & & \times & \times & \times & \\
& \times & & & \times & & \times
\end{pmatrix}$$

(a) (b)

Figure 1: A 7×7 naturally ordered matrix and its counterpart ordered by nested dissection ordering.

$$\begin{pmatrix}
1 & \times & & & \times & & & \\
\times & 2 & & & \times & & & \\
& & 3 & \times & & & & \\
& & \times & 4 & \times & \times & \times & \\
\times & \times & & \times & 5 & & & \times \\
& & & \times & & 6 & \times & \\
& & & \times & & \times & 7 & \times \\
& & & & \times & & \times & 8
\end{pmatrix}$$

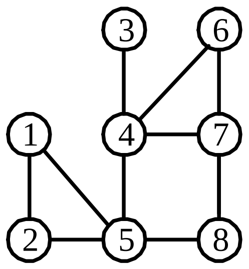

Figure 2: An 8×8 matrix and its associated graph.

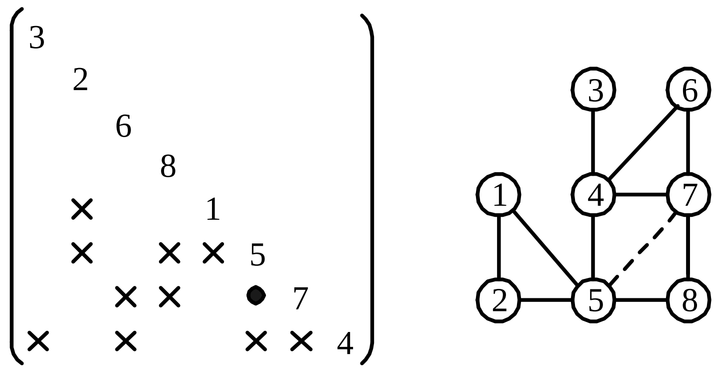

Figure 3: The Cholesky factor of the matrix in Figure 2 ordered by 3 2 6 8 1 5 7 4 and the associated filled graph.

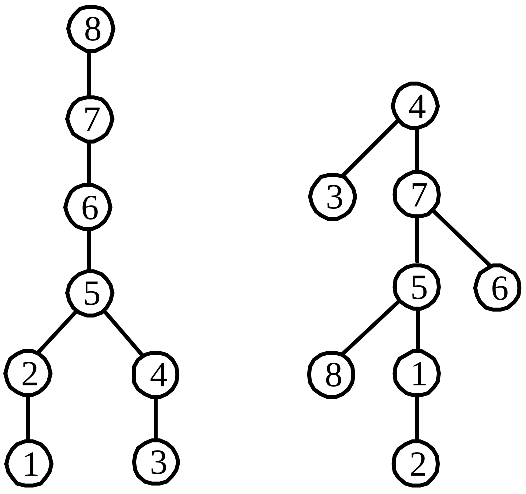

Figure 4: The elimination trees of the matrix in Figure 2 that correspondingly ordered by natural ordering and 3 2 6 8 1 5 7 4.

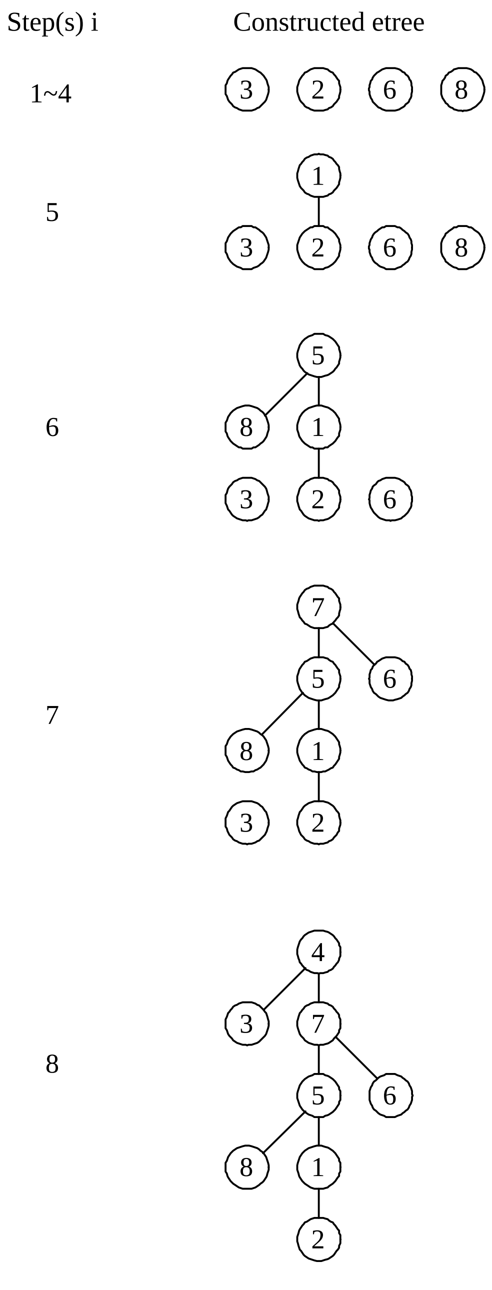

Figure 5: Stages in executing Algorithm 2 on the graph in Figure 2 ordered by 3 2 6 8 1 5 7 4.

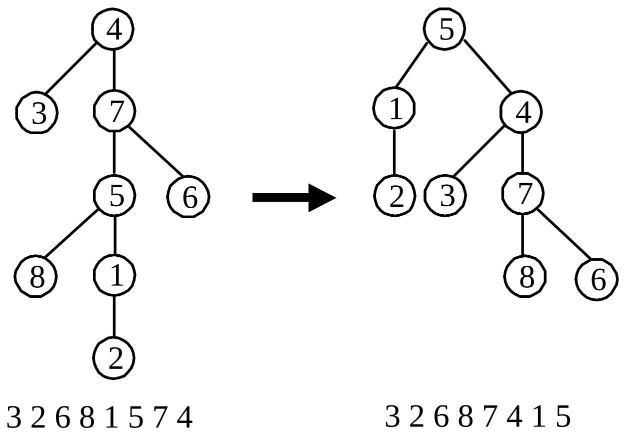

Figure 6: The corresponding elimination trees of the matrix in Figure 2 before and after a tree rotate mutation.

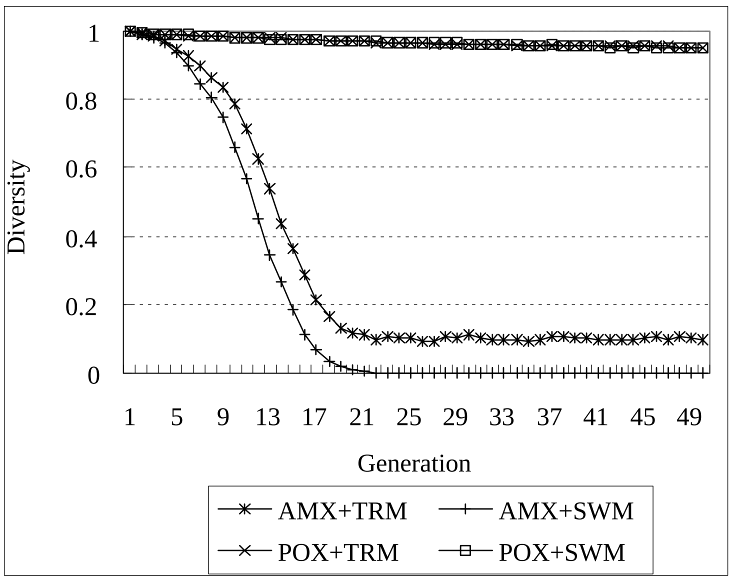

Figure 7: Diversity vs. generation, for matrix BCSPWR09 ordered by MMD.

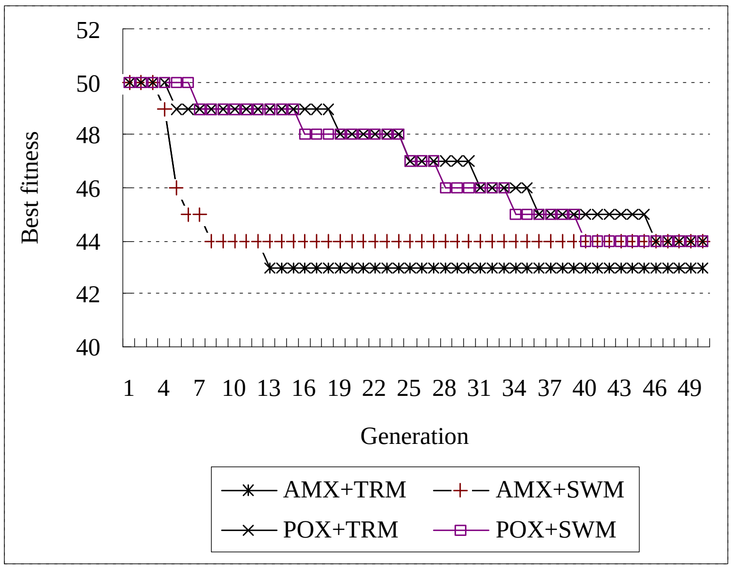

Figure 8: Best results achieved vs. generation, for matrix BCSPWR09 ordered by MMD.

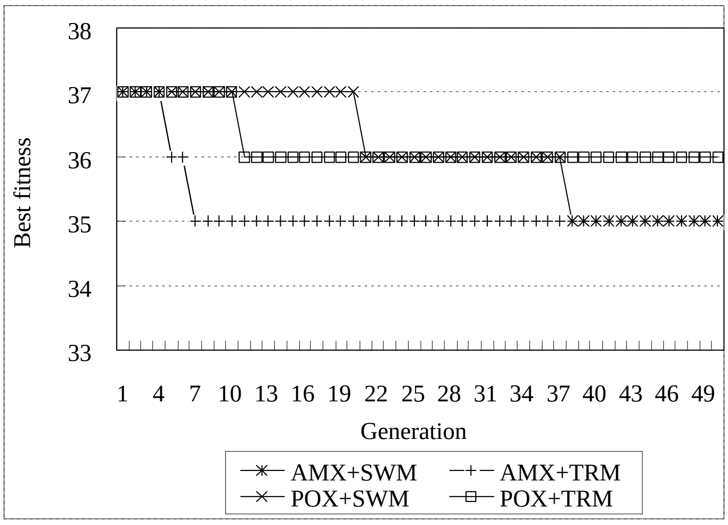

Figure 9: Best results achieved vs. generation, for matrix BCSPWR09 ordered by METIS.

Table 1: Test matrices from the Harwell-Boeing sparse matrix collection.

Key	Order	Nonzeros
BCSPWR01	39	85
BCSPWR02	49	108
BCSPWR03	118	297
BCSPWR04	274	943
BCSPWR05	443	1033
BCSPWR06	1454	3377
BCSPWR07	1612	3718
BCSPWR08	1624	3837
BCSPWR09	1723	4117

Table 2: Improvement of the proposed GA in reducing elimination tree height, compared with MMD+JK.

Key	MDD+JK	GA	Imprv	Gen
BCSPWR01	7	6	14%	5
BCSPWR02	8	6	25%	8
BCSPWR03	14	12	14%	4
BCSPWR04	38	33	13%	6
BCSPWR05	25	22	14%	6
BCSPWR06	39	34	13%	13
BCSPWR07	41	36	12%	13
BCSPWR08	43	37	14%	8
BCSPWR09	50	43	14%	12
Average			15%	8

Table 3: Improvement of the proposed GA in reducing elimination tree height, compared with METIS+JK.

Key	METIS+JK	GA	Imprv	Gen
BCSPWR01	7	7	–	–
BCSPWR02	6	6	–	–
BCSPWR03	13	12	8%	3
BCSPWR04	33	30	9%	6
BCSPWR05	23	22	4%	4
BCSPWR06	32	30	6%	6
BCSPWR07	33	31	6%	4
BCSPWR08	37	35	5%	6
BCSPWR09	42	41	2%	5
Average			6%	5