A Multilevel Unsymmetric Matrix Ordering Algorithm for Parallel Process Simulation

Y. F. Hu[*], K. C. F. Maguire and R. J. Blake

Daresbury Laboratory, Daresbury,

Warrington WA4 4AD, United Kingdom

Abstract

Computer simulation of complex chemical processes is increasingly being used in the design, optimization and control of chemical facilities. Industrial-scale modeling involves the solution of large systems of algebraic differential equations. This is very computationally intensive with a large part of the computing time attributed to the repeated solution of large, sparse, unsymmetric systems of linear equations. One way of speeding up the simulation is to solve the linear systems efficiently in parallel by reordering the unsymmetric matrices into a bordered block-diagonal (BBD) form. In this paper a multilevel ordering algorithm is presented. A multilevel technique, which provides a global view, is combined with a Kernighan-Lin algorithm to form an effective unsymmetric matrix ordering algorithm – MONET (Matrix Ordering for minimal NET-cut). Numerical results confirm that this algorithm gives ordering of better quality than existing algorithms.

[*]Author to whom all correspondence should be addressed. Tel. +44 (0)1925 603683; Fax +44 (0)1925 603634; e-mail: y.f.hu@dl.ac.uk

Keywords chemical process simulation, sparse unsymmetric matrix ordering, bordered block diagonal, graph partitioning, multilevel, parallel computation.

1 Introduction

Computer simulation and optimization of complex chemical processes are increasingly used in the design and control of chemical facilities. Industrial-scale simulation involves the solution of large systems of algebraic differential equations, which is very computationally intensive. In (Zitney et al., 1995), a dynamic simulation of Bayer AG required 18 hours of CPU time on a Cray C90 supercomputer using the standard implementation of SPEEDUP (Aspen Technology, Inc.). In most of these simulations, the majority of the computing time is spent in the repeated solution of large, sparse, unsymmetric linear systems resulting from the application of Newton's method to nonlinear systems. In order to reduce the solution time for such linear systems, a number of studies have investigated the exploitation of computational parallelism. For example, frontal and multi-frontal linear solvers have been developed to better exploit fine-grained parallelism in vector and shared memory parallel architectures (Duff & Reid, 1983; Duff & Reid, 1984; Amestoy & Duff, 1989; Zitney & Stadtherr, 1993; Zitney et al., 1995; Zitney et al., 1996; Mallya & Stadtherr, 1997). In (Zitney et al., 1995), the use of a frontal solver, combined with improvements in other aspects including I/O performance, have reduced the run time to 21 minutes for the afore-mentioned simulation.

Very high performance computers are usually based on distributed memory architectures, or a combination of shared and distributed memory architectures. Effective parallel computers, consisting of a network of workstations or PCs, are also becoming increasingly popular in academia as well as in industry. Distributed memory parallel computers require algorithms that exploit coarse-grain parallelism, because of the relatively higher cost of interprocessor memory/data access. A number of such parallel algorithms for the solution of symmetric positive definite systems have been proposed (Gupta et al., 1997).

For unsymmetric systems arising in process simulation, coarse-grain parallel algorithms can be developed by reordering the matrices into a bordered block-diagonal (BBD) form, and subsequently factorizing the diagonal blocks independently (Vegeais & Stadtherr, 1992; Coon & Stadtherr, 1995; Mallya et al., 1997a; Mallya et al., 1997b). This strategy is suitable because a matrix from process simulation can occur naturally in this form, if explicit knowledge of the unit-stream nature of the problem is applied in the formulation of the matrix (Westerberg & Berna, 1978). On the other hand, because of the disparity of process units and thus the sizes of the diagonal blocks, load balancing can become a problem. Furthermore, in commercial software, information about the unit-structure may not be known to the sparse matrix solver. Rather, the solver is presented with a general sparse matrix, which is normally highly unsymmetric, with no desirable structural or numerical properties such as diagonal dominance or narrow bandedness. This makes it difficult to design robust and efficient parallel iterative algorithms (Cofer & Stadtherr, 1996). Instead, a parallel sparse direct solver coupled with an automatic reordering of the matrix becomes more appropriate.

It has been demonstrated that the performance of a parallel sparse direct solver based on the BBD form depends strongly on the quality of the reordering (Mallya et al., 1997a; Mallya et al., 1997b), measured in terms of the size of the border (the net-cut) and the load balance of the diagonal blocks (the uniformity of the size of the diagonal blocks). Given that systems of the same structure are solved repeatedly in process simulation, a good quality reordering represents a one-off effort that could have substantial long-term pay-backs (Mallya et al., 1997a; Mallya et al., 1997b). Therefore this paper will focus on the development of an efficient, high quality ordering algorithm.

There are existing algorithms for ordering unsymmetric matrices into bordered block-diagonal form, without explicit knowledge of the underlying physical problem. The MNC (Min-Net-Cut) algorithm (Coon & Stadtherr, 1995) starts with a matrix of zero-free diagonal (full transversal), and employs row and column exchanges that maintain a full transversal. The TPABLO algorithm (Choi & Szyld, 1996) is a simple algorithm designed for structurally symmetric matrices. This algorithm has been applied to the symmetrized form $(A + A^T)$ of structurally unsymmetric matrices, but was found to be unsatisfactory (Mallya et al., 1997a; Mallya et al., 1997b).

Recently Camarda and Stadtherr (Camarda & Stadtherr, 1998) simplified the MNC algorithm. It was argued that the preservation of the structural nonsingularity of the diagonal blocks is not necessary, because a pivoting strategy is normally applied during the factorization of the diagonal blocks. Any entries that are not eliminated can be moved to the border to form the interface problem. Their algorithm, GPA-SUM (Graph-Partitioning Algorithm for Sparse Unsymmetric Matrices), which performs row exchanges without the constraint of maintaining a zero-free diagonal, was demonstrated to result in bordered block-diagonal matrices with smaller border size, more diagonal blocks and better load balance.

Both MNC and GPA-SUM are recursive bisection algorithms based on the Kernighan-Lin (KL) algorithm (Kernighan & Lin, 1970). GPA-SUM does not allow a row exchange unless it decreases the net-cut. MNC follows the KL algorithm more strictly by permitting moves with negative gain, in the hope of getting out of local minima.

The KL algorithm is known to be a greedy local optimization algorithm. Allowing moves with negative gain may increase the chance of getting out of local minima to some extent, nonetheless the quality of the final bisection has been shown to depend greatly on the quality of the initial bisection (Hu & Blake, 1994). In recent years multilevel techniques, which provide a global view of the problem, have been combined with the local steepest decent nature of the KL algorithm to form efficient and high quality (undirected) graph partitioning algorithms (Barnard & Simon, 1994; Hendrickson & Leland, 1993b; Karypis & Kumar, 1995; Karypis & Kumar, 1998a; Karypis & Kumar, 1998b; Karypis & Kumar, 1999; Oliker & Biswas, 1998; Walshaw et al., 1995; Walshaw et al., 1997). These algorithms can also be used to order symmetric matrices into BBD form, even though most of the graph partitioning algorithms minimize the edge-cut of the graphs, rather than the net-cut of the matrix – a more important measure that is directly linked with the border size. It is noted that one of these graph partitioning algorithms, METIS (Karypis & Kumar, 1998a), has an option for the number of shared vertices to be minimized, which makes it more suitable for minimizing net-cut when used to order symmetric matrices into BBD form, compared with edge-cut based graph partitioning algorithms.

This paper presents a multilevel algorithm for the ordering of unsymmetric matrices into BBD form. Given the great advantages of the multilevel approach, it is surprising that there has been little published work on the application of this approach to the unsymmetric matrix ordering problem, probably because of the difficulties in handling asymmetry. Two exceptions are the works in (Çatalyürek & Aykanat, 1996) and (Hendrickson & Kolda, 1998), which were brought to the authors attention after the completion of the work presented in this paper. The works in (Çatalyürek & Aykanat, 1996) and (Hendrickson & Kolda, 1998) are concerned with minimizing the communication time in parallel sparse matrix vector multiplications, which differs from our motivation of minimizing the border size of BBD matrices. Two hyper-graph models were used in (Çatalyürek & Aykanat, 1996), whilst a bipartite graph model was used in (Hendrickson & Kolda, 1998), and in both cases the edge-cut was minimized. On the other hand, our work is based on the row connectivity graphs and the Galerkin operator, with the objective of minimizing the net-cut. Our work also differs from (Çatalyürek & Aykanat, 1996; Hendrickson & Kolda, 1998) in that a matrix coarsening strategy based on heavy edge collapsing is adopted, guided by the strength of row connectivity.

The main contribution of this paper is in the successful application of the multilevel approach to unsymmetric matrices, combined with a KL algorithm for minimizing net-cut. The resulting algorithm is demonstrated to give orderings into the bordered block-diagonal form, with much smaller border size and better load balance than existing algorithms. New algorithms which apply undirected graph partitioning algorithms to the row connectivity graph have also been investigated.

In section 2, the problem of ordering unsymmetric matrices into bordered block-diagonal form is defined. Existing algorithms are described in further detail. Section 3 presents the authors' implementation of the KL algorithm, as well as the motivation and design of the multilevel ordering algorithm MONET. In Section 4, MONET is compared with some new algorithms which apply existing graph partitioning softwares to row connectivity graphs, as well as the two existing algorithms MNC and GPA-SUM, and is found to be of better quality. Section 5 discusses future work.

2 The Matrix Ordering Problem

The (single) bordered block-diagonal matrices of concern in this paper are of the form

$$\begin{pmatrix} A_{11} & & & & S_1 \\ & A_{22} & & & S_2 \\ & & \ddots & & \vdots \\ & & & A_{NN} & S_N \end{pmatrix}. \tag{1}$$

where the A_{ii} are $m_i \times n_i$ matrices, in general rectangular with $m_i \geq n_i$. The basic idea of the parallel implementation is given below, further details can be found in (Mallya et al., 1997a; Mallya et al., 1997b).

Using N processors, each processor i has one diagonal block A_{ii} and part of the border S_i held locally in its memory. The diagonal blocks are then factorized in parallel, with row or column pivots chosen within each block. Entries of the diagonal blocks that cannot be eliminated, including those not eliminated for numerical reasons, are then combined with the border elements to form a double bordered block-diagonal matrix

$$\begin{pmatrix} L_1U_1 & & & & U_1^{'} \\ & L_2U_2 & & & U_2^{'} \\ & & \ddots & & \vdots \\ & & & L_NU_N & U_N^{'} \\ L_1^{'} & L_2^{'} & \dots & L_N^{'} & F \end{pmatrix}. \tag{2}$$

Here F is the interface matrix which contains contributions from the factorization of each diagonal block, in the standard Schur complement fashion. The interface variables are then solved. Finally the remaining variables can be solved in parallel using the factorization on each processor.

In the above process, the factorization of the interface matrix F is essentially a sequential process, and therefore a possible bottleneck. This interface matrix is likely to be dense and is thus more costly to factorize than a sparse matrix of the same size, though there may be scope to solve it in parallel using a parallel dense solver such as ScaLapack. It is therefore important to keep the size of the interface matrix to a minimum so as to achieve a good overall speedup of the solution process. Furthermore, due to load balancing considerations, the diagonal blocks should preferably be of similar size, under the assumption that the number of floating point calculations per block is approximately proportional to its size (see Section 4 for further discussion).

Of course the matrices presented to a linear solver are usually not ordered in the above form, and those that are may not be load balanced. Thus there is a need for an automatic matrix ordering algorithm.

For structurally symmetric matrices, there are many available algorithms, which are in fact undirected graph partitioning algorithms. Of particular note are the Kernighan-Lin algorithm (Fiduccia & Mattheyses, 1982; Kernighan & Lin, 1970), the spectral bisection algorithm (Simon, 1991) and various multilevel schemes (Barnard & Simon, 1994; Hendrickson & Leland, 1993b; Karypis & Kumar, 1995; Karypis & Kumar, 1998a; Karypis & Kumar, 1998b; Karypis & Kumar, 1999; Oliker & Biswas, 1998; Simon et al., 1998; Walshaw et al., 1995; Walshaw et al., 1997). Continuous development, stimulated by the need to parallelize unstructured mesh based applications, has resulted in a number of powerful graph partitioning codes, the most widely used of which include Chaco (Hendrickson & Leland, 1993a) and, more recently, METIS (Karypis & Kumar, 1998a) and JOSTLE (Walshaw et al., 1995).

It is possible to make use of these undirected graph partitioning algorithms to order an unsymmetric matrix A into BBD form. One way to achieve this is to apply the graph partitioning algorithm to $A + A^T$. However it has been shown (Coon & Stadtherr, 1995; Mallya et al., 1997a; Mallya et al., 1997b), and will again be demonstrated in Section 4, that such a strategy is likely to offer an ordering of low quality for highly unsymmetric matrices. As rightly argued by Coon and Stadtherr (1995), this is because if A is highly unsymmetric, this approach misses the parallelism that exists in A, but not in $A + A^T$, due to the fact that the undirected graph $A + A^T$ contains data dependencies that do not exist in the original system A. The authors believe that another reason for the ineffectiveness of this strategy is that by applying an undirected graph partitioning algorithm to $A + A^T$, one is restricted to symmetric permutations of the matrix A, which represent only a small percentage of all possible (symmetric and unsymmetric) permutations. Furthermore, graph partitioning algorithms usually attempt to minimize the edge-cut of the graph, rather than the net-cut of the matrix – a more important measure. Algorithms that are specifically designed to order unsymmetric matrices are therefore necessary and can be expected to give ordering of better quality.

2.1 Nets and net-cut

The structure of a sparse *symmetric* matrix A can be represented by an undirected graph $G = (V, E)$, where V is the set of vertices (row and column indices) and E the set of edges. An edge between vertex i and j exists if $a_{ij} \neq 0$, where a_{ij} is the entry of the matrix at row i and column j. Note that by symmetry, if $a_{ij} \neq 0$, then $a_{ji} \neq 0$, so there is no direction associated with the edges.

The structure of a sparse *unsymmetric* matrix A on the other hand can be represented by a directed graph $\vec{G} = (V, \vec{E})$. An edge from vertex i to j exists if $a_{ij} \neq 0$. Notice that this does not imply that the edge $j \to i$ exists.

Definition 1 *A* NET *(Coon & Stadtherr, 1995) is defined as a set of indices related to a column of a sparse matrix. It consists of all the row indices of the nonzero entries in that column.*

For example, the net corresponding to column 4 of the matrix in Figure 1 is

$\{1, 4, 5, 7\}$. A *net* is *cut* by a partition of the matrix, if two of its row indices belong to different parts of the partition. For example in Figure 2, nets 2, 3, 4, 6 and 8 are cut by the partition (illustrated by the broken line), whereas nets 1, 5 and 7 are not.

Definition 2 *The* NET-CUT *of a partition is defined as the number of nets that are cut by that partition.*

For instance, the net-cut by the partition shown in Figure 2 is 5.

In the ideal case when a partition has zero net-cut, the matrix can be reordered into block-diagonal form with no border at all. Although this is unlikely for a general matrix, the aim of the ordering algorithm is to reorder unsymmetric matrices so that the partition gives as small a net-cut as possible. At the same time, each partition should have approximately the same number of rows so as to maintain load balance. To achieve this, the following concept is useful.

Definition 3 *The* GAIN *associated with moving a row of a partitioned matrix into another partition is the reduction in the net-cut that will result after such a move. The gain is negative if such a move increases the net-cut.*

On the right hand side of Figure 1, the gain for each row, with respect to the "natural" initial partitioning indicated by the broken line, is given.

2.2 The row connectivity graph

The matrix ordering algorithm of this paper utilizes the concepts of row connectivity and row connectivity graphs, first introduced by Mayoh (1965). They are defined as follows.

Definition 4 *Two rows of a matrix are* CONNECTED *to each other, if they share one or more column indices.*

Definition 5 *A* ROW CONNECTIVITY GRAPH G_A *of a matrix* A *is a graph consisting of the row indices of* A *as vertices. Two vertices* i *and* j *of the graph are connected by an edge if rows* i *and* j *of the matrix are connected. The number of shared column indices between row* i *and row* j *of the matrix is called the* EDGE WEIGHT *of the edge* (i, j) *of the graph, and the number of nonzero entries in row* i *of the matrix is called the* VERTEX SIZE *of vertex* i *of the graph*

For example, in Figure 1, rows 3 and 4 are not connected, while rows 3 and 6 are connected with an edge weight of 3. The vertex sizes of rows 3 and 6 are 4 and 3, respectively.

It has been proved that the structure of the row connectivity graph G_A is given by the matrix product AA^T (Tewarson, 1973). In fact the following new result with regard to the edge weights and vertex sizes can be proved. The proof is relatively straightforward and is thus not given here.

Theorem 1 *Let* A *be an unsymmetric matrix of dimension* m *by* n. *Let* $\bar{A}$ *be the matrix* A *but with all nonzero entries set to 1. Then vertices* i *and* j $(i \neq j)$ *form an edge of the row connectivity graph* G_A *if and only if* $(\bar{A}\bar{A}^T)_{ij} \neq 0$. *The edge weight of this edge is* $(\bar{A}\bar{A}^T)_{ij}$. *The vertex size of the vertex* i *of graph* G_A *equals* $(\bar{A}\bar{A}^T)_{ii}$.

Due to Theorem 1, hereafter G_A will be used to denote the row connectivity graph of A, as well as the matrix $\bar{A}\bar{A}^T$.

As an example, for the matrix A in Figure 1, the matrix representation of its row connectivity graph is

$$\bar{A}\bar{A}^T = \begin{pmatrix} 5 & 1 & 1 & 4 & 5 & & 4 & 1 \\ 1 & 1 & 1 & & 1 & & 1 & 1 \\ 1 & 1 & 4 & & 1 & 3 & 1 & 4 \\ 4 & & & 4 & 4 & & 3 & \\ 5 & 1 & 1 & 4 & 5 & & 4 & 1 \\ & & 3 & & & 3 & & 3 \\ 4 & 1 & 1 & 3 & 4 & & 4 & 1 \\ 1 & 1 & 4 & & 1 & 3 & 1 & 4 \end{pmatrix} \tag{3}$$

Here the off-diagonal values correspond to the edge weights of the row connectivity graph. For instance entry $(3,4)$ is zero (rows 3 and 4 of A are not connected), while entry $(3,6)$ is 3 (rows 3 and 6 of A have 3 common column indices). The values on the diagonal correspond to the vertex sizes of the row connectivity graph. For example, entry $(1,1)$ is 5 because row 1 of A has 5 nonzero entries.

Definition 6 *A row in partition i is said to be on the* BOUNDARY *between partitions i and j, if it is connected to a row in partition j. If a row is not on the boundary, it is said to be an* INTERIOR *row.*

For example, row 2 in Figure 2 is on the boundary, because it is connected with rows 3, 5, 7 and 8 in the partition below the broken line. On the other hand, row 6 is an interior row since it is not connected with any other rows in the partition above the broken line.

3 The Multilevel Matrix Ordering algorithm

The matrix ordering algorithm proposed in this paper is based on the multilevel approach. Given the original unsymmetric matrix, a series of matrices will be generated, each *coarser* (with smaller row dimension) than the preceding one. The coarsest matrix is then bisected. This bisection is prolonged to the finer matrices and refined using the KL algorithm. Partitioning of matrices into more than two blocks can be achieved by recursive bisection.

In this section the implementation of the KL refinement algorithm is described first, followed by the multilevel approach.

3.1 The KL refinement algorithm

The KL algorithm (Kernighan & Lin, 1970) was first suggested in 1970 for bisecting undirected graphs in relation to VLSI circuit layout. It is an iterative algorithm. Starting from a load balanced initial bisection, it first calculates the gain for each vertex. Here the gain of a vertex is defined as the reduction of edge-cut (the number of edges of the undirected graph cut by the bisection) that results, if that vertex is moved from one partition of the graph to the other. At each inner iteration, the algorithm moves the vertex which has the highest gain, from the partition in surplus (that is, the partition with more vertices) to the partition in deficit. This vertex is then locked and the vertex gains updated. The procedure is repeated even if the highest gain is negative, until all of the vertices are locked. The last few moves that have negative gains are then undone and the bisection is reverted to the one with the smallest edge-cut so far in this iteration. This completes one outer iteration of the KL algorithm, and the iterative procedure is restarted. Should an outer iteration fail to result in any reductions in the edge-cut, the algorithm is terminated. The initial bisection is generated randomly and for large graphs, the final result is very dependent on the initial choice. The KL algorithm is a local optimization algorithm, with a limited capability of getting out of local minima by way of allowing moves with negative gain.

By using appropriate data structures, it is possible to implement the KL algorithm so that each outer iteration has a complexity of $O(|E|)$ (Fiduccia & Mattheyses, 1982), where $|E|$ denotes the number of edges in the undirected graph.

The principle of this algorithm was applied in the context of the ordering of unsymmetric matrices by Coon and Stadtherr (Coon & Stadtherr, 1995). They started with a matrix having a zero-free diagonal, and considered permutations of row and column vertices of a bipartite graph representation of the matrix. Row and column indices that were permuted were locked for the rest of an outer iteration. Two types of moves were identified which may reduce net-cut without destroying the zero-free diagonal. The first move permutes both row and column i to the other partition; the second move finds a cycle on the bipartite graph consisting of unlocked row and column vertices, and permutes the corresponding row indices. The second move can be potentially expensive to identify. Restricting the cycle to a length of four was found to be a good compromise.

Because the above algorithm (referred to as MNC in (Camarda & Stadtherr, 1998)) needs to preserve a zero-free diagonal, the moves are rather restrictive. Camarda and Stadtherr (Camarda & Stadtherr, 1998) sought to simplify the algorithm by not maintaining the zero-free diagonal. They argued that structural non-singularity does not guarantee numerical non-singularity, and since a pivoting strategy must be used anyway, starting with structurally non-singular matrices is not strictly necessary. Instead, they used row permutations only, with no restriction except that of load balance, so as to allow a greater number of possible row moves. The algorithm, GPA-SUM, was again a recursive bisection algorithm. The rows were sorted by their gains. Rows that were moved are locked for the rest of this outer iteration. Two types of moves were considered. The first exchanges unlocked rows between partitions, the second moves a single row into the other partition. In GPA-SUM only moves that decrease the net-cut are allowed. The resulting algorithm GPA-SUM was found to give smaller net-cut, better load balance and more blocks when compared with the MNC algorithm. It was also found to be faster (Camarda & Stadtherr, 1998).

Our implementation of the KL algorithm follows more closely that of GPA-SUM in the sense that only row permutations are considered. However it differs from GPA-SUM in two respects. The first is that rows with negative gains may also be moved in the hope of escaping local minima. Such a move is called the uphill move, formally defined as follows.

Definition 7 *Given a row partition of a matrix, the move of a row from partition p to partition q is said to be* UPHILL*, if such a move increase either the net-cut, or the local load imbalance. Here* LOCAL LOAD IMBALANCE *is defined as the absolute value of the difference of the number of rows in partitions p and q.*

The second difference in our implementation of the KL algorithm to that of GPA-SUM is that the row vertex with the highest gain is moved, one at a time, from the partition in deficit to the partition in surplus. Row exchange is not employed because the gain of a row exchange is more difficult to calculate than a simple sum of two individual gains, since moving one row vertex may affect the gain of the other.

Following the practice in the partitioning of undirected graphs (Karypis & Kumar, 1999; Walshaw et al., 1995) of reducing the computational complexity of the KL algorithm, only rows on the boundary are considered for moving. Here we recall, from Definition 6, that a row is on the boundary if it is connected to a row in another partition. We also follow Karipis & Kumar (1999) by setting a limit on the number of uphill moves (moves which increase the net-cut). The proposed KL algorithm for bisecting unsymmetric matrices is as follows, where two partitions, 0 and 1, will result.

Algorithm KL

- *1. Set the initial bisection. Initialize the gain of each row on the boundary*
- *2. do $ITER = 1$,* MAX_OUT_ITER
  - *2.1. choose from the partition in surplus (in case of a tie, chose from partition 0) an unlocked boundary row i with the highest gain; move to the other partition.*
  - *2.2. if the previous move is an uphill move, and the number of consecutive uphill moves exceed* MAX_UPHILL_STEPS*, or if all rows are locked, undo all the uphill moves, unlock all rows and goto Step 2.4*
  - *2.3. lock row i, update the gains and the boundary, repeat from Step 2.1.*
  - *2.4. if net-cut or load balance is not improved, exit.*
- *end do*

Notice that the number of continuous uphill moves is restricted to MAX_UPHILL_STEPS, so as to reduce the computational complexity of the KL algorithm further. The trade-off is that the algorithm is more likely to get stuck in local minima if MAX_UPHILL_STEPS is set too small. For an efficient implementation, only the gains of rows connected to the moved row are updated, because the gains of the remaining rows are not affected. Gains of boundary rows are kept in a linked list of gain-buckets. Rows with the same gain are inserted into the same bucket. This allows efficient retrieval and insertion of gains (Fiduccia & Mattheyses, 1982; Karypis & Kumar, 1999)

Figures 1-10 illustrate the KL algorithm on a randomly generated 8×8 matrix. This matrix is shown in Figure 1, with an initial net-cut of 8 corresponding to the partitioning illustrated with the broken line. At the beginning, both partitions have the same number of rows and to break the tie, a row is chosen to move from partition 0 (top half of the matrix). It is seen from Figure 1 that the highest gain in partition 0 is achieved at row 3, because moving this row will eliminate the net-cut at columns 2, 5 and 7. After moving that row to partition 1 (bottom half of the matrix), the gains are updated as in Figure 2 and the net-cut becomes $8 - 3 = 5$. Now partition 1 is in surplus and the highest gain in this partition is at row 5. Moving row 5 to partition 0 results in Figure 3 with a net-cut of 4. Next row 2 is moved from partition 0 to partition 1 to break the tie in load balance, which gives the matrix in Figure 4. Now partition 1 is in surplus and row 7 is moved to partition 0 with a gain of 3, the resulting matrix in Figure 5 has a net-cut of 1. The rest of moves up to Figure 9 are all uphill moves and at Figure 9 all rows are locked. The uphill moves are undone and the KL outer iteration can restart from Figure 10. For this particular example, no improvement in net-cut and load balance can be achieved in the next KL outer iteration and the final matrix in Figure 10 can be ordered by moving column 3 to the border, and rearrange the other columns. The reordered matrix is shown in Figure 11 with the border part marked by "B".

3.2 The Multilevel Approach

A multilevel approach, in the context of the partitioning of undirected graphs (Barnard & Simon, 1994; Karypis & Kumar, 1998a; Walshaw et al., 1995), generates a series of coarser and coarser graphs. Each coarse graph encapsulates the information needed to partition the original graph from its "parent", but contains fewer vertices. The coarsest graph is partitioned. This graph has very few degrees of freedoms, and can therefore be partitioned rapidly. The partitions on the coarse graphs are recursively prolonged (usually by injection) to the finer graphs, with further refinement at each level.

One of the first uses of the multilevel approach for the partitioning of undirected graph was reported in (Barnard & Simon, 1994), where the multilevel approach is combined with the spectral bisection algorithm, mainly to reduce the partitioning time. It was soon realized (Hendrickson & Leland, 1993b; Karypis & Kumar, 1999; Walshaw et al., 1995) that the multilevel approach provides a global view of the problem, and therefore complements ideally the KL algorithm which is a good local optimizer.

There are three main phases in the application of the multilevel approach – coarsening, partitioning and prolongation. Details of how these three phases are implemented in our multilevel algorithm for unsymmetric matrices are discussed in the following.

3.2.1 The coarsening phase

There are a number of ways to coarsen an undirected graph. In (Barnard & Simon, 1994), the *maximal independent set* of a graph is chosen as the seed vertices for the coarse graph. An independent set of a graph is a set of its vertices, with no two of the vertices in the set connected by an edge of the graph. An independent set is a maximal independent set if the addition of an extra vertex makes it no longer independent.

As an alternative, in (Walshaw et al., 1995), a greedy algorithm was used to form small clusters of, say, 10 vertices for collapsing. This allows more aggressive coarsening.

However, the most popular method for generating the multilevel hierarchy of coarse undirected graphs is based on edge-collapsing (Hendrickson & Leland, 1993b; Karypis & Kumar, 1999). Selected edges are collapsed so that the two vertices connected by one of these edges forms a multi-node. Each vertex of the resulting coarse graph has a weight associated with it, indicating the number of original vertices that it represents. Each edge of the coarse graph also has a weight associated with it, indicating the number of original edges that it represents. The edges to be collapsed are usually selected using the idea of *matching*. A matching of a graph is a set of edges, such that no two of them share the same vertex. The maximal match is a matching of the largest possible size.

In the context of partitioning of undirected graphs, *heavy edge matching* (Karypis & Kumar, 1999) has been found to work well. Vertices are visited randomly. For each of these vertices, the unmatched edge from the vertex with the highest edge weight is selected. Heavy edge matching has the advantage that the resulting coarse graph has a relatively small total edge weight, and therefore partitioning it is more likely to give a small edge-cut (the sum of the weights of the edges cut by a partitioning) (Karypis & Kumar, 1999).

Another advantage of the edge-collapsing approach in undirected graph partitioning is that the edge-cut on the coarsest graph equals that on the finest (original) graph, if the partitioning on the coarsest graph were injected to the finer graphs without further refinement. Therefore to some extent the original problem is encapsulated well by problems of smaller sizes. For this reason, in our unsymmetric matrix ordering algorithm, the idea of edge-collapsing is also used.

Our strategy for "coarsening" an unsymmetric matrix is to match two rows of the matrix if they are strongly connected. Here two rows are strongly connected if the corresponding edge in the row connectivity graph has a large edge weight. All row vertices are initially unlocked. An edge in the row connectivity graph is said to be a free edge if its two end vertices are unlocked. Unlocked row vertices are visited, and the free edge that is connected to such a vertex with the highest edge weight is selected. The two end row vertices are then locked. This process is repeated for the remaining unlocked row vertices until all rows are locked.

Matched rows of the matrix are finally collapsed. The rationale for this heavy edge collapsing strategy is that if two rows are strongly connected, they must have many identical column entries and should be kept on the same partition, so as to avoid a high net-cut.

The *weight* of a row vertex, defined as the number of rows of the original matrix that this row vertex represents, is accumulated. The resulting matrix has fewer rows but the same number of columns. The coarsening process is applied recursively until either the number of rows in the coarse matrix is less than the pre-set minimal number (say 100), or the ratio of the number of rows between the fine and coarse matrices is over a certain constant. The latter is necessary for the following reasons. On some matrices, after many levels of coarsening, the coarsest matrix has one super row vertex of very high vertex weight, often higher than 50% of the total vertex weight. In such a case, subsequent coarsening will not reduce the number of rows significantly. Furthermore, the presence of a super vertex in the next two phases of the algorithm, namely partitioning and prolongation, may create load imbalance. Therefore coarsening will stop if the ratio of the number of rows between two subsequent matrices exceeds 0.8.

3.2.2 The partitioning phase

Because the coarsest matrix has few rows, it can be bisected quickly with high quality by many algorithms. To ensure good load balance, the bisection is done such that the summations of the row vertex weights for each sub-matrix are roughly the same.

The choice of the partitioning algorithm at this coarsest level is not very important to the overall quality of the partitioning because refinement will be carried out at the subsequent finer levels. This was also found to be the case in undirected graph partitioning (Hendrickson & Leland, 1993b). We have tested both the KL algorithm with MAX_UPHILL_STEPS = ∞, which allows the KL algorithm unlimited uphill steps in the hope of finding the global minimum net-cut, and also the METIS (Karypis & Kumar, 1998a) graph partitioning code (applied to the row connectivity graph). It was found that both give similar results. The former is therefore used by default.

3.2.3 The prolongation phase

During the prolongation phase, the partitioning of a coarse matrix is inherited by the fine matrix through simple injection. The resulting partition on the fine matrix is further refined using the KL algorithm. Because the fine matrix inherited a good starting partition, the size of the boundary should only be a small fraction of that of the interior. Therefore the KL algorithm, which takes the boundary refinement approach, should be quite efficient.

3.2.4 The two level algorithm

With all the three main phases described, the complete multilevel algorithm can be formulated. As is standard practice with multi-grid type algorithms, it is sufficient to give only the 2-level algorithm. The superscript f and c will be used to represent the fine and coarse matrix quantities respectively. For example A^f will be the fine matrix of size $m_f \times n_f$ and A^c will be the matrix after coarsening, of size $m_c \times n_c$. The following notion of a *bisector* is needed.

Definition 8 *A* BISECTOR *x corresponding to the bisection of an $m \times n$ matrix A is a vector of size m, consisting of zeros and ones. $x_i = 0$ (respectively, 1) if row i of the matrix belongs to partition 0 (respectively, 1).*

For example the bisector corresponding to the bisection shown in Figure 3 is $x = (1, 1, 0, 1, 1, 0, 0, 0)$. The 2-level algorithm is now given as follows.

The two level algorithm

- *Coarsening phase:*
  - *form the row connectivity matrix* $G_A^f = \bar{A}^f (\bar{A}^f)^T$.
  - *form the prolongation operator* P *and the restriction operator* $R = P^T$.
  - *form the coarse matrix* $A^c = RA^f$ *and coarse matrix row connectivity graph* $G_A^c = \bar{A}^c (\bar{A}^c)^T$
- *Partitioning phase:*
  - *Use the KL algorithm with* MAX_UPHILL_STEPS $= \infty$ *to bisect* A^c, *starting with the natural initial natural bisector* $x_0^c = (0, 0, \ldots, 0, 1, \ldots, 1)$. *The resulting bisector is* x^c.
- *Prolongation phase:*
  - *On the fine matrix, starting with the initial bisector* $x_0^f = Px^c$, *apply the KL refinement algorithm to give the final bisector* x^f.

The prolongation operator is the operator that gives the fine grid initial bisector from the coarse grid bisector, that is,

$$x_0^f = Px^c.$$

In the case of injection combined with heavy edge-collapsing as the coarsening strategy, this prolongation operator is a sparse matrix of size $m_f \times m_c$, defined as

$$P_{ij} = \begin{cases} 1, & \text{if row vertex } i \text{ of } A_f \text{ collapes into row vertex } j \text{ of } A_c \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The coarse matrix is formed by the row restriction of the fine matrix, $A^c = RA^f$, where $R = P^T$ is the restriction operator. It could have been formed using the full Galerkin operator with the same restriction operator but with a prolongation operator chosen such that heavily connected columns are collapsed together. This would give

$$A^c = RA^f \tilde{P},$$

with $\tilde{P}$ chosen independent of R. This approach would reduce both the column and row dimensions. However such a formulation would not preserve the nice property that the net-cut of the coarse matrix equals that of the fine matrix after injection. Since this property contributes to the high quality of the multilevel approach, row restriction only is carried out in this work.

Notice that the coarse row connectivity graph could also have been formed directly through the Galerkin operator $G_A^c = RG_A^f P$. However such a formulation does not provide the correct edge weights. For this reason the coarse row connectivity graph is generated directly from the coarse matrix.

The above 2-level algorithm is applied recursively to form the multilevel algorithm. More specifically, the partitioning phase of the 2-level algorithm is replaced by another 2-level algorithm. This recursive nesting gives a multilevel algorithm where the original matrix is coarsened many times (until no further coarsen can be carried out, as outlined at the end of Section 3.2.1). The coarsest matrix is then partitioned. This is followed by the prolongation and refinement of the partition to finer matrices.

Figure 12 illustrates the result of applying the multilevel algorithm to bisect a matrix of size 381×381. On the left-hand side, three levels of coarsening are carried out. The coarsest matrix, on the bottom left, is of size 58×381. The partitioned coarsest matrix is shown on the bottom right (in the right hand part of Figure 12, columns are reordered so as to show clearly the bordered block-diagonal form, in practice this column reordering is not carried out). This matrix is then prolonged to the finer levels and refined there. At the finest level, the reordered original matrix (top right of Figure 12) is shown in the bordered block-diagonal form, with 2 diagonal blocks and a net-cut of 51 (in percentage terms this is 13.39%).

4 Numerical Results

The software based on the above multilevel algorithm is named MONET (Matrix Ordering for minimal NET-cut). It is available from the corresponding author.

In this section, MONET is compared with other algorithms for a range of test matrices. Table 1 lists these matrices in alphabetical order, with a description of the size of the matrices and their sources. The matrices are either from the Harwell-Boeing Collection (http://www.dci.clrc.ac.uk/Activity/SparseMatrices) or from the University of Florida Sparse Matrices Collection (http://www.cise.ufl.edu/∼davis).

In MONET, by default the maximum number of levels in the multilevel algorithm is set to MAX_LEVEL $= \infty$, and parameters for the KL refinement algorithm in the prolongation phase are set to MAX_UPHILL_STEPS = 100 and MAX_OUT_ITER = 10. These parameters are chosen rather arbitrarily. They are not specially tuned for the test problems.

A few variations of the algorithm were also tested. These variations, together with the default algorithm, are listed as follows:

- MONET – the default multilevel KL algorithm with MAX_LEVEL $= \infty$, MAX_UPHILL_STEPS = 100 and MAX_OUT_ITER = 10.
- $\mathrm{MONET_S}$ – a single level KL algorithm with MAX_LEVEL = 1, MAX_UPHILL_STEPS = 100 and MAX_OUT_ITER = 10.
- $\mathrm{MONET_S^A}$ – a single level aggressive KL algorithm with MAX_LEVEL = 1,

MAX_UPHILL_STEPS = ∞ and MAX_OUT_ITER = 1000.

- MONET$^{\mathrm{A}}_{\mathrm{M}}$ – a multilevel aggressive KL algorithm. MAX_LEVEL = ∞, MAX_UPHILL_STEPS = ∞ and MAX_OUT_ITER = 1000.

MONET$_{\mathrm{S}}$ is the single level KL algorithm with a moderate ability to climb out of a local minima. By comparing MONET with MONET$_{\mathrm{S}}$, we hope to see the benefit of using the multilevel approach. MONET$^{\mathrm{A}}_{\mathrm{S}}$ and MONET$^{\mathrm{A}}_{\mathrm{M}}$ are the aggressive version of the single and multilevel algorithms. Here the KL algorithm is allowed to perform as many uphill steps as necessary, in the hope that a better approximation to the global minimum can be reached, at the expense of higher computational cost.

One of the simplest ways of ordering an unsymmetric matrix A is to apply a symmetric matrix ordering algorithm to $A + A^T$. However this approach was found to be unsatisfactory (Mallya et al., 1997a; Mallya et al., 1997b), as shall be further illustrated later. A better approach is to reorder the rows of the matrix based on the row connectivity matrix, $G_A = \bar{A}\bar{A}^T$. The rows are partitioned by using a graph partitioning algorithm that minimizes the edge-cut on G_A. It can be proved that the edge-cut of a partitioning of G_A is an upper bound of the net-cut of the same row partitioning of A. The resulting ordering can be further improved using the KL algorithm. Therefore the following two additional new algorithms are tested

- MET – applying the graph partitioning algorithm METIS (Karypis & Kumar, 1998a), software available from http://www-users.cs.umn.edu/~karypis /metis/metis.html) to G_A. The version used for this paper was METIS 3.00.

- MetMo — partition using METIS, further refinement using MONET$_{\mathrm{S}}$ (the single level KL algorithm).

It should be stressed that METIS is a very efficient and high quality software package for graph partitioning based on minimizing edge-cut. In using it for ordering unsymmetric matrices, which is concerned with minimizing net-cut, we are not testing the quality of METIS. Therefore comparison figures on net-cut should not be viewed as a measure of the quality of METIS. Rather, using METIS on the symmetric matrix G_A is a simple way to order an unsymmetric matrix, and as such is a benchmark that any new ordering algorithm for unsymmetric matrices should measure itself against and exceed.

The quality and efficiency of the algorithms are measured by

- *net-cut* – Here the net-cut is expressed in percentage terms, which is the net-cut divided by the order of the matrix. The smaller the net-cut, the better the quality of the algorithm.

- *load imbalance* – This is defined (Camarda & Stadtherr, 1998) as the difference between the largest block size and the average block size, normalized by the average block size, expressed as a percentage. The smaller the load imbalance, the better.

- CPU time.

Note that the work needed to factorize a block is related to the number of non-zeros as well as the sparse structure of the block. A better measure of the load imbalance is therefore the variation in the expected number of floating-point operations needed to factorize each of the blocks. However this is more difficult to calculate *a priori* and the above measure, which only takes into account the size of the diagonal blocks, is used as a crude but practical approximation.

The algorithms were applied to the 29 test problems on a Digital computer with a 300 MHz Alpha EV5 processor. The net-cut and load imbalance are listed in Table 2 for four blocks and in Table 3 for sixteen blocks. The CPU times are listed in Tables 4 and 5. All these tables are divided into two vertical regions, with MONET, MET and MetMo on the left, and the variants of MONET on the right. In the left hand region of Tables 2 and 3, the lowest net-cut for each problem (among the three algorithms MONET, MET and MetMo) is highlighted.

As can be seen from Tables 2 and 3, all our algorithms gave small load imbalances except on very small problems, where a minor variation in the block size can cause a large change in the load imbalance.

We first compare MONET with its variants in terms of the net-cut and the CPU time. For most cases, MONET gave a net-cut similar to those given by the single level aggressive KL algorithm $\mathrm{MONET}_{\mathrm{S}}^{\mathrm{A}}$ and the multilevel aggressive KL algorithm $\mathrm{MONET}_{\mathrm{M}}^{\mathrm{A}}$, indicating that MONET possesses the same capability of getting out of local minima as the aggressive versions of the KL algorithm, while only requiring a limited number of uphill moves. It is therefore computationally less expensive, as seen from Tables 4 and 5. As expected, $\mathrm{MONET}_{\mathrm{M}}^{\mathrm{A}}$ gives the lowest net-cuts on most problems, due to the combination of the multilevel technique and the aggressive KL algorithm. This is however achieve at the cost of much higher CPU times. Overall, MONET gave high quality orderings with relatively low CPU times. This algorithm is therefore taken as a good compromise between quality and efficiency, and is used as the default.

We now compare MONET with MET and MetMo. In terms of the net-cut, the multilevel algorithm MONET is generally better than MET. MONET is also better than MetMo, a hybrid of the METIS and KL algorithms, for 19 out of the 29 test cases when the matrices are partitioned into 4 blocks, and for 18 out of 29 test cases when the matrices are partitioned into 16 blocks. The underperformance occurs mostly for small test problems.

In terms of CPU times (Tables 4 and 5), the single level KL algorithm

$\mathrm{MONET_S}$ is the least expensive, followed by MET and MetMo. The timings for MET include that of the METIS 3.00 software itself, as well as that of forming the G_A matrix. For example, when the largest problem (lhr71) is partitioned into 16 blocks, 39 of the 89.89 seconds required by MET are spent in METIS 3.00 itself, with the rest of the time spent mainly in forming the row connectivity matrix G_A. MONET is roughly 1.4 times slower than MET. This is because on each level, the row connectivity matrix has to be formed, which is computationally expensive. MONET takes twice the time needed by the single level algorithm $\mathrm{MONET_S}$, which indicates that the computational cost is reduced by roughly a factor of two as MONET moves from one level to the corresponding coarse level in the multilevel hierarchy. This is achieved with the help of the merging of supervariables – columns of *exactly* the same pattern, a technique used frequently in sparse matrix ordering (e.g., (Reid & Scott, 1998)) for computational efficiency.

It is interesting to compare the two simple approaches to ordering – that of using $A+A^T$ and that of using the row connectivity graph ($G_A = \bar{A}\bar{A}^T$), combined with the undirected graph partitioning software METIS. Through experimenting on a number of cases, it was found that the latter (MET) was always much better. This is possibly due to the fact that all the matrices in the experiment are highly unsymmetric. To give some typical examples, it was found that when bisecting the lhr04 test case, MET gives a net-cut of 3.05%, while the $A + A^T$ approach gives a net-cut of 18.95%. On the hydr1 example, MET gives a net-cut of 0.64%, while the $A + A^T$ approach gives 14.6%.

Table 6 compares MONET with two existing algorithms, GPA-SUM (Camarda & Stadtherr, 1998) and MNC (Coon & Stadtherr, 1995). The lowest net-cuts are highlighted. The net-cut and load imbalance figures for GPA-SUM and MNC were taken from (Camarda & Stadtherr, 1998), except those for the two largest test cases (lhr71 and lhr34), which were taken from (Coon & Stadtherr, 1995) since results were not available from (Camarda & Stadtherr, 1998).

In (Camarda & Stadtherr, 1998), it was found that GPA-SUM was similar or better than MNC in terms of smaller net-cuts as well as better load balance. From Table 6 it is evident that in general, MONET gives much smaller load imbalances. The load imbalances for MNC are especially large. On the lhr10 case the imbalance is 151.6%! In terms of net-cut, MONET is better than both GPA-SUM and MNC for most of the test cases. This is particularly true on the matrices from the Harwell-Boeing Collections, where MONET sometimes gives a net-cut 5 times smaller than GPA-SUM. On the lhr series of test cases, the advantage of MONET in terms of net-cut is not as great. For the two largest cases, MONET gives slightly worse net-cut than MNC, but we suspect this is due to the extra number of diagonal blocks (the number of diagonal blocks for MONET is currently restricted to a power of two, while for MNC the number of blocks may not be a power of two). MONET however gives far smaller load imbalance.

5 Discussion and Future Work

In this paper a multilevel ordering algorithm MONET for unsymmetric matrices has been proposed. In terms of net-cut and load balance, the MONET algorithm has been demonstrated to be of much higher quality than two existing algorithms. It has also been found to be of better quality compared to two new algorithms Met and MetMo, which apply the METIS graph partition software to the row connectivity graph.

At present, MONET restricts the number of partitions to a power of two. This restriction however is easily removed by bisecting into sub-matrices of unequal row weight sums, as in (Hu & Blake, 1994). The real challenge for the ordering algorithm could however be on cases where the matrix has an underlying block diagonal form with say 10 blocks and the block sizes vary around a mean. In such a case, there are two difficulties

- The ordering algorithm has no *a priori* information on the number of blocks. For this example 10 may be the best number of partitions.
- The ordering algorithm has no *a priori* information on the amount of imbalance that should be allowed, with the trade-off that strict load balance may cause higher net-cut.

Although all the numerical results in Section 4 are derived by requiring strict load balance, MONET does allow a load imbalance tolerance to be set at each KL refinement step. MONET was tested on randomly generated bordered block diagonal matrices with 10 blocks. The size of each block was allowed to vary by 10% from the mean. These matrices were randomly permuted to conceal the BBD form. It was found that by allowing some load imbalance, it is possible to recover exactly the original bordered block diagonal form. However it is difficult to know in advance how much load imbalance should be allowed. Therefore it may be necessary to build a model for the computational time of the parallel solution of a linear system in BBD form, as a function of the number of blocks, the interface size and the load imbalance. The ordering algorithm should then minimize that cost function.

It would be interesting to take into account the values of the nonzeros in the matrix in the ordering process, so that the resulting ordered matrix is numerically more desirable. The authors also hope to extend the bisection algorithm to direct k-way partitioning.

The motivation of this paper is to solve, in parallel, general unsymmetric systems arising from chemical process simulation, through the reordering of such systems into a bordered block-diagonal form. MONET represents an ordering algorithm which gives high quality ordering and with reasonable efficiency. This make it possible to solve these linear systems much faster in parallel, which in turn opens up the possibility of more realistic and complex physical models, as well as shorter solution times for existing simulations. To this end work is underway on combining MONET with a direct sparse solver to form a parallel direct solver, or to be used as a parallel preconditioner for iterative algorithms.

Acknowledgments

The authors would like to thank Dr. Jennifer Scott and Prof. Ian Duff for many valuable comments and suggestions on a draft of this paper.

References

Amestoy, P. R. & Duff, I. S. (1989). Vectorization of a multiprocessor multifrontal code. *International Journal of Supercomputer Applications and High Performance Computing*, **3**, 41–59.

Barnard, S. T. & Simon, H. D. (1994). Fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, **6**, 101–117.

Camarda, K. V. & Stadtherr, M. A. (1998). Matrix ordering strategies for process engineering: graph partitioning algorithms for parallel computation. Computers & Chemical Engineering, to be published.

Çatalyürek, U. V. & Aykanat, C. (1996). Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. *Lecture Notes in Computer Science*, **1117**, 75–86.

Choi, H. & Szyld, D. B. (1996). Application of threshold partitioning of sparse matrices to markov chains. Technical report 96-21, Dept. of Mathematics, Temple Univ., Philadephia, PA. Available from `http://www.math.temple.edu/~szyld`.

Cofer, H. N. & Stadtherr, M. A. (1996). Reliablity of iterative linear solvers in chemical process simulation. *Computers & Chemical Engineering*, **20**, 1123–1132.

Coon, A. B. & Stadtherr, M. A. (1995). Generalized block-tridiagonal matrix orderings for parallel computation in-process flowsheeting. *Computers & Chemical Engineering*, **19**, 787–805.

Duff, I. S. & Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear-equations. *ACM Transactions on Mathematical Software*, **9**, 302–325.

Duff, I. S. & Reid, J. K. (1984). The multifrontal solution of unsymmetric sets of linear-equations. *SIAM Journal on Scientific and Statistical Computing*, **5**, 633–641.

Fiduccia, C. M. & Mattheyses, R. M. (1982). A linear time heuristic for improving network partitions. In *Proc. 19th ACM-IEEE Design Automation Conf.*, Las Vegas, ACM.

Gupta, A., Karypis, G., & Kumar, V. (1997). Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, **5**, 502–520.

Hendrickson, B. & Kolda, T. G. (1998). Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel processing. SIAM Journal of Scientific Computing, to be published. Available from http://www.cs.sandia.gov/∼bahendr/papers.html.

Hendrickson, B. & Leland, R. (1993a). The chaco user's guide, version 1.0. Technical report sand93-2339, Sandia National Laboratories, Allbuquerque, NM.

Hendrickson, B. & Leland, R. (1993b). A multilevel algorithm for partitioning graphs. Technical report sand93-1301, Sandia National Laboratories, Allbuquerque, NM.

Hu, Y. F. & Blake, R. J. (1994). Numerical experiences with partitioning of unstructured meshes. *Parallel Computing*, **20**, 815–829.

Karypis, G. & Kumar, V. (1995). Parallel multilevel graph partitioning. Technical report, Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

Karypis, G. & Kumar, V. (1998a). Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, **48**, 96–129.

Karypis, G. & Kumar, V. (1998b). A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, **48**, 71–95.

Karypis, G. & Kumar, V. (1999). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, **20**, 359–392.

Kernighan, B. W. & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell Systems Tech. J.*, **49**, 291–308.

Mallya, J. U. & Stadtherr, M. A. (1997). A multifrontal approach for simulating equilibrium-stage processes on supercomputers. *Industrial & Engineering Chemistry Research*, **36**, 144–151.

Mallya, J. U., Zitney, S. E., Choudhary, S., & Stadtherr, M. A. (1997a). A parallel block frontal solver for large scale process simulation: Reordering effects. *Computers & Chemical Engineering*, **21**, s439–s444.

Mallya, J. U., Zitney, S. E., Choudhary, S., & Stadtherr, M. A. (1997b). A parallel frontal solver for large-scale process simulation and optimization. *AICHE JOURNAL*, **43**, 1032–1040.

Mayoh, B. H. (1965). A graph technique for inverting certain matrices. *Mathematics of Computation*, **19**, 198–213.

Oliker, L. & Biswas, R. (1998). Plum: Parallel load balancing for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, **52**, 150–177.

Reid, J. K. & Scott, J. A. (1998). Ordering symmetric sparse matrices for small profile and wavefront. Technical report ral-tr-98-016, Rutherford Appleton Laboratory, Oxon OX11 0QX, UK.

Simon, H. D. (1991). Partitioning of unstructured problems for parallel processing. *Computer Systems in Engineering*, **2**, 135–148.

Simon, H. D., Sohn, A., & Biswas, R. (1998). Harp: A dynamic spectral partitioner. *Journal of Parallel and Distributed Computing*, **50**, 83–103.

Tewarson, R. P. (1973). *Sparse Matrices*. Academic Press.

Vegeais, J. A. & Stadtherr, M. A. (1992). Parallel processing strategies for chemical process flowsheeting. *AICHE Journal*, **38**, 1399–1407.

Walshaw, C., Cross, M., & Everett, M. (1995). Dynamic mesh partitioning: a unified optimisation and load-balancing algorithm. Technical report 95/im/06, University of Greenwich, London SE18 6PF, UK.

Walshaw, C., Cross, M., & Everett, M. G. (1997). Parallel dynamic graph partitioning for adaptive unstructured meshes. *Journal of Parallel and Distributed Computing*, **47**, 102–108.

Westerberg, A. W. & Berna, T. J. (1978). Decomposition of very large-scale newton-raphson based flowsheeting problems. *Computers & Chemical Engineering*, **2**, 61.

Zitney, S. E., Brull, L., Lang, L., & Zeller, R. (1995). Plantwide dynamic simulation on supercomputers - modeling a bayer distillation process. *AIChE Symp. Ser.*, **91**, 313–316.

Zitney, S. E., Mallya, J., Davis, T. A., & Stadtherr, M. A. (1996). Multifrontal vs frontal techniques for chemical process simulation on supercomputers. *Computers & Chemical Engineering*, **20**, 641–646.

Zitney, S. E. & Stadtherr, M. A. (1993). Frontal algorithms for equation-based chemical process flowsheeting on vector and parallel computers. *Computers & Chemical Engineering*, **17**, 319–338.

Table 1: The test matries, their order, number of non-zeros in the matrix and source

	Problem	Order	Non-zeros	Source
1	lhr71	70304	1528092	Florida
2	lhr34	35152	764014	Florida
3	lhr17	17576	381975	Florida
4	lhr14	14270	307858	Florida
5	lhr11	10964	233741	Florida
6	lhr10	10672	232633	Florida
7	lhr07	7337	156508	Florida
8	hydr1	5308	23752	Florida
9	lhr04	4101	82682	Florida
10	lns_3937	3937	25407	Harwell-Boeing
11	sherman5	3312	20793	Harwell-Boeing
12	lhr02	2954	37206	Florida
13	rdist3a	2398	61896	Florida
14	oilgen	2205	14133	Harwell-Boeing
15	west2021	2021	7353	Harwell-Boeing
16	west1505	1505	5445	Harwell-Boeing
17	lhr01	1477	18592	Florida
18	mahindas	1258	7682	Florida
19	gre_1107	1107	5664	Harwell-Boeing
20	west0989	989	3537	Harwell-Boeing
21	bp_1000	822	4661	Florida
22	west0655	655	2854	Harwell-Boeing
23	west0497	497	1727	Harwell-Boeing
24	west0479	479	1910	Harwell-Boeing
25	west0381	381	2157	Harwell-Boeing
26	west0167	167	507	Harwell-Boeing
27	west0156	156	371	Harwell-Boeing
28	west0132	132	414	Harwell-Boeing
29	west0067	67	294	Harwell-Boeing

Table 2: Percentage of net-cut/load imbalance for six algorithms to order the test problems into bordered block diagonal form with 4 blocks

problem	MONET	MET	MetMo	$\mathrm{MONET_S}$	$\mathrm{MONET_S^A}$	$\mathrm{MONET_M^A}$
lhr71	**0.45**/0.00	0.67/0.01	0.68/0.00	0.27/0.00	0.27/0.00	0.27/0.00
lhr34	**1.62**/0.00	2.41/0.01	2.37/0.00	1.53/0.01	1.49/0.01	1.05/0.01
lhr17	**2.58**/0.02	4.25/0.02	3.92/0.02	2.48/0.02	2.57/0.02	2.18/0.66
lhr14	**2.77**/2.65	3.19/0.01	4.74/0.01	3.41/0.01	2.95/0.01	2.77/2.65
lhr11	**3.81**/0.11	5.69/0.00	5.26/0.04	4.10/0.00	4.07/0.00	3.41/0.00
lhr10	**3.60**/0.04	5.53/0.04	5.13/0.04	5.15/0.04	3.85/0.00	3.17/1.09
lhr07	**4.48**/0.86	7.46/0.04	4.51/0.04	7.13/0.04	6.38/0.04	4.48/0.86
hydr1	2.00/8.59	1.94/0.08	**1.53**/0.00	2.90/0.00	2.98/0.08	1.43/0.00
lhr04	**7.75**/0.07	10.49/0.07	9.68/0.07	9.12/0.07	9.12/0.07	8.34/0.07
lns_3937	**9.98**/0.08	10.01/0.08	**9.98**/0.08	10.11/0.08	10.08/0.08	9.98/0.08
sherman5	**7.88**/0.00	13.74/0.00	12.50/0.00	20.02/0.00	9.06/0.00	6.97/0.00
lhr02	**6.64**/0.07	11.00/0.20	10.66/0.07	9.55/0.07	9.55/0.07	6.64/0.07
rdist3a	**7.59**/0.08	11.18/0.08	12.59/0.08	8.72/0.08	7.67/0.08	7.59/0.08
oilgen	**17.96**/0.14	18.23/0.14	18.10/0.14	19.41/0.14	20.36/0.14	18.10/0.14
west2021	7.67/0.15	5.10/0.15	**3.02**/0.15	15.83/0.15	8.86/0.15	4.95/0.15
west1505	4.92/0.20	7.84/0.20	**4.52**/0.20	16.01/0.20	8.97/0.20	4.19/0.20
lhr01	**12.32**/0.20	13.34/0.20	13.27/0.20	14.83/0.20	14.83/0.20	12.32/0.20
mahindas	**19.16**/0.16	25.12/0.16	21.62/0.16	19.71/0.16	19.32/0.16	18.52/0.16
gre_1107	**25.75**/0.45	32.70/0.09	28.18/0.09	27.10/0.09	27.37/0.09	25.75/0.45
west0989	9.50/0.71	6.98/0.30	**5.46**/0.30	16.08/0.30	9.00/0.30	9.30/0.71
bp_1000	**26.03**/0.24	29.81/0.24	29.44/0.24	26.64/0.24	24.70/0.24	26.03/0.24
west0655	**12.06**/0.76	31.76/0.15	23.36/0.15	14.05/0.15	12.98/0.15	10.23/0.15
west0497	**8.25**/0.60	36.62/0.60	10.26/0.60	12.07/0.60	8.85/0.60	8.25/0.60
west0479	16.70/0.21	17.12/1.88	**15.24**/0.21	15.66/0.21	16.70/0.21	16.49/0.21
west0381	41.73/0.79	48.82/0.79	**40.68**/0.79	46.98/0.79	40.94/0.79	40.68/0.79
west0167	19.76/2.99	17.96/7.78	**10.78**/2.99	14.37/0.60	10.18/0.60	10.78/0.60
west0156	14.74/0.00	**12.18**/2.56	**12.18**/0.00	16.03/0.00	12.82/0.00	11.54/0.00
west0132	17.42/0.00	27.27/6.06	**15.91**/0.00	15.15/0.00	15.91/0.00	17.42/0.00
west0067	52.24/1.49	55.22/7.46	**50.75**/7.46	52.24/1.49	52.24/1.49	52.24/1.49

Table 3: Percentage of net-cut/load imbalance for six algorithms to order the test problems into bordered block diagonal form with 16 blocks

problem	MONET	MET	MetMo	$\mathrm{MONET_S}$	$\mathrm{MONET_S^A}$	$\mathrm{MONET_M^A}$
lhr71	**2.92**/0.02	4.75/0.02	4.24/0.02	3.11/ 0.02	3.05/ 0.02	2.44/ 0.66
lhr34	**6.36**/2.69	7.23/0.09	8.54/0.05	6.50/ 0.05	6.18/ 0.05	5.19/ 1.59
lhr17	**9.82**/2.05	11.69/0.05	10.47/0.14	10.94/ 0.14	9.22/ 0.05	8.00/ 1.78
lhr14	**9.80**/2.70	11.24/0.13	12.98/0.13	13.24/ 0.13	11.07/ 0.13	9.36/ 2.70
lhr11	**13.55**/1.28	16.65/0.26	17.06/0.11	15.24/ 0.11	12.72/ 0.11	11.81/ 0.11
lhr10	**13.91**/2.70	17.28/0.30	16.00/0.15	16.09/ 0.15	12.77/ 0.15	14.53/ 1.20
lhr07	**19.38**/2.06	24.83/0.31	20.06/0.31	20.34/ 0.31	20.32/ 0.31	18.37/ 2.06
hydr1	7.08/8.82	8.20/0.38	**6.57**/0.38	9.97/ 0.08	7.82/ 0.08	6.35/ 0.08
lhr04	33.04/0.27	34.97/0.66	**30.77**/0.27	33.02/ 0.27	31.80/ 0.27	30.87/ 0.27
lns_3937	**29.03**/0.38	30.86/0.38	30.84/0.38	34.39/ 0.38	31.12/ 0.38	30.15/ 0.38
sherman5	**21.44**/0.00	29.95/0.00	27.26/0.00	37.77/ 0.00	27.72/ 0.00	19.60/ 0.00
lhr02	**26.10**/0.74	32.94/0.74	30.43/0.74	31.99/ 0.20	31.65/ 0.20	26.17/ 0.74
rdist3a	39.78/0.08	**33.32**/4.09	35.32/0.75	41.53/ 0.75	40.95/ 0.75	39.78/ 0.08
oilgen	**45.58**/0.86	49.30/0.86	46.80/0.14	49.93/ 0.14	45.71/ 0.14	43.27/ 0.14
west2021	11.53/1.34	18.01/0.54	**10.34**/1.34	18.65/ 0.54	14.35/ 0.54	10.39/ 2.13
west1505	12.76/1.00	34.82/1.00	**12.03**/5.25	17.94/ 1.00	15.28/ 1.00	12.16/ 1.00
lhr01	**40.62**/0.74	51.59/1.83	44.21/0.74	49.22/ 0.74	48.54/ 0.74	39.95/ 0.74
mahindas	**24.32**/6.84	37.76/1.75	27.03/1.75	27.74/ 1.75	24.01/ 0.48	24.48/ 1.75
gre_1107	**60.79**/1.17	67.84/1.17	61.25/1.17	64.95/ 1.17	62.06/ 1.17	59.62/ 1.17
west0989	15.67/1.92	25.88/3.54	**12.64**/10.01	20.53/ 0.30	15.98/ 1.92	16.89/ 1.92
bp_1000	**49.03**/1.22	54.87/3.16	51.22/1.22	50.00/ 1.22	45.50/ 1.22	49.03/ 1.22
west0655	**25.19**/2.60	47.79/2.60	36.18/2.60	25.80/ 2.60	25.04/ 2.60	24.12/ 2.60
west0497	**21.33**/3.02	49.50/9.46	21.53/9.46	22.54/ 3.02	23.14/ 3.02	21.33/ 3.02
west0479	**31.32**/3.55	41.34/6.89	32.78/3.55	29.65/ 3.55	30.06/ 3.55	30.48/ 3.55
west0381	77.69/4.99	94.49/9.19	**75.59**/9.19	75.59/ 4.99	72.70/ 4.99	76.38/ 4.99
west0167	32.34/5.39	49.10/24.55	**28.14**/14.97	31.14/14.97	29.34/ 5.39	30.54/ 5.39
west0156	27.56/12.82	25.64/12.82	**22.44**/2.56	25.64/ 2.56	25.64/12.82	25.64/ 2.56
west0132	35.61/9.09	66.67/9.09	**31.82**/9.09	34.85/ 9.09	34.09/ 9.09	35.61/ 9.09
west0067	88.06/19.40	88.06/19.40	**83.58**/19.40	88.06/19.40	88.06/19.40	88.06/19.400

Table 4: CPU time (in seconds) for six algorithms to order the test problems into bordered block diagonal form with 4 blocks

problem	MONET	MET	MetMo	$\mathrm{MONET_S}$	$\mathrm{MONET_S^A}$	$\mathrm{MONET_M^A}$
lhr71	56.29	41.93	51.89	23.97	185.82	739.57
lhr34	27.01	20.95	28.71	12.66	321.53	380.15
lhr17	13.46	10.61	11.97	7.26	187.80	196.58
lhr14	10.72	8.26	11.14	4.97	108.68	148.99
lhr11	9.20	6.68	7.48	5.27	73.08	137.59
lhr10	9.31	6.81	7.46	5.64	53.29	104.35
lhr07	7.46	4.57	6.41	3.62	50.97	87.33
hydr1	0.94	0.55	0.60	0.25	2.33	7.78
lhr04	4.19	2.17	2.75	1.88	18.10	51.90
lns_3937	0.75	0.49	0.65	0.46	3.69	7.66
sherman5	2.87	0.79	0.86	0.30	2.84	5.32
lhr02	1.28	0.75	1.18	0.41	3.68	15.60
rdist3a	1.92	1.39	1.56	0.69	12.60	18.74
oilgen	0.49	0.25	0.38	0.31	2.28	3.78
west2021	0.43	0.17	0.26	0.09	0.91	1.33
west1505	0.25	0.13	0.18	0.07	0.67	0.96
lhr01	0.76	0.38	0.57	0.25	2.43	5.82
mahindas	1.31	0.35	0.66	0.19	5.08	7.47
gre_1107	0.32	0.13	0.21	0.10	0.65	1.00
west0989	0.19	0.09	0.14	0.04	0.49	0.47
bp_1000	0.30	0.09	0.14	0.09	0.62	1.03
west0655	0.14	0.08	0.11	0.03	0.20	0.38
west0497	0.13	0.08	0.09	0.03	0.21	0.34
west0479	0.09	0.04	0.08	0.03	0.11	0.22
west0381	0.24	0.10	0.19	0.08	0.62	0.76
west0167	0.02	0.02	0.03	0.01	0.03	0.04
west0156	0.01	0.02	0.02	0.01	0.02	0.02
west0132	0.02	0.01	0.02	0.01	0.02	0.03
west0067	0.01	0.02	0.03	0.01	0.01	0.02

Table 5: CPU time (in seconds) for six algorithms to order the test problems into bordered block diagonal form with 16 blocks

problem	MONET	MET	MetMo	$\mathrm{MONET_S}$	$\mathrm{MONET_S^A}$	$\mathrm{MONET_M^A}$
lhr71	110.95	89.89	96.77	52.14	943.47	1454.40
lhr34	57.69	47.28	54.81	23.96	569.73	768.49
lhr17	32.20	26.99	25.53	15.24	279.24	308.03
lhr14	27.02	16.42	22.12	9.68	188.47	310.23
lhr11	25.40	12.09	12.85	9.38	135.21	259.29
lhr10	18.44	13.77	16.93	7.33	114.46	165.41
lhr07	11.45	7.65	8.37	7.50	83.40	133.43
hydr1	1.45	0.89	1.30	0.61	4.95	8.04
lhr04	6.95	5.22	6.66	3.54	38.08	75.81
lns_3937	1.96	1.39	1.76	0.94	6.70	11.10
sherman5	3.83	1.13	1.73	0.61	5.30	6.99
lhr02	4.16	2.73	2.92	0.93	8.76	22.53
rdist3a	4.23	2.89	3.26	1.51	22.12	31.55
oilgen	1.14	0.89	0.59	0.52	3.61	5.21
west2021	0.66	0.33	0.38	0.18	1.12	2.14
west1505	0.54	0.29	0.29	0.12	0.85	2.27
lhr01	2.38	0.76	0.98	0.55	4.41	9.54
mahindas	1.51	0.66	0.73	0.41	5.23	5.32
gre_1107	0.56	0.28	0.32	0.18	1.01	1.55
west0989	0.28	0.17	0.34	0.08	0.60	1.10
bp_1000	0.43	0.29	0.43	0.14	0.82	2.44
west0655	0.17	0.14	0.29	0.07	0.26	1.27
west0497	0.16	0.14	0.20	0.06	0.27	0.43
west0479	0.12	0.16	0.16	0.05	0.16	0.46
west0381	0.30	0.30	0.35	0.12	0.71	1.02
west0167	0.03	0.05	0.11	0.02	0.05	0.07
west0156	0.02	0.04	0.07	0.02	0.03	0.03
west0132	0.03	0.05	0.07	0.02	0.03	0.04
west0067	0.02	0.03	0.04	0.02	0.02	0.03

Table 6: Comparing MONET with GPA-SUM and MNC. The entries give the number of blocks/net-cut (%)/load imbalance (%)

problem	MONET	GPA-SUM	MNC
lhr71	16/2.92/0.02	-	10/**2.1**/31.1
lhr34	8/3.23/0.02	-	6/**2.2**/57.2
lhr17	4/**2.58**/0.02		4/6.90/9.3
lhr17	8/**5.83**/2.05	8/7.44/6.3	-
lhr17	16/**9.82**/2.05	16/11.8/8.2	-
lhr14	8/**5.31**/2.70	8/9.01/6.0	6/5.40/58.8
lhr14	16/**9.80**/2.70	16/13.4/10.6	-
lhr11	8/**7.45**/0.84	8/9.87/6.9	10/9.64/148.8
lhr11	16/**13.55**/1.28	16/16.3/11.8	-
lhr10	8/**7.19**/0.07	8/9.59/6.0	10/9.65/151.6
lhr10	16/**13.91**/2.70	16/15.8/7.9	-
lhr07	8/**10.09**/0.86	8/11.7/7.0	-
lhr07	16/19.38/2.06	16/**18.1**/11.0	-
hydr1	4/**2.00**/8.59	-	4/3.43/8.8
hydr1	8/**3.86**/8.67	8/9.19/9.9	-
hydr1	16/**7.08**/8.82	16/12.9/18.5	-
lhr04	4/7.75/0.07	4/7.85/3.4	3/**6.91**/49.6
lhr04	8/16.51/0.07	8/**15.9**/10.2	-
lns_3937	2/**3.33**/0.03	2/21.7/10.0	-
lns_3937	4/**9.98**/0.08	4/55.2/10.2	6/18.0/72.2
sherman5	2/**4.26**/0.00	2/18.8/10.1	-
sherman5	4/**7.88**/0.00	4/41.8/16.7	4/22.7/105.4
west2021	2/**5.54**/0.05	2/15.0/15.3	3/9.25/70.6
west2021	16/**11.53**/1.34	16/21.9/24.4	-
west1505	2/**2.06**/0.07	2/15.1/1.0	3/9.17/62.7
west1505	16/**12.76**/1.00	16/22.7/9.5	-
mahindas	2/**14.94**/0.00	2/25.0/8.3	-
mahindas	4/**19.16**/0.16	4/35.8/29.8	-
gre_1007	2/12.65/0.09	2/34.8/3.5	2/**11.8**/5.9
west0989	2/**5.66**/0.1	2/15.5/1.1	3/10.4/59.6
west0989	16/**15.67**/1.92	15/23.4/9.2	-
bp_1000	2/15.45/0.00	2/20.2/6.3	2/**14.1**/19.0
bp_1000	4/**26.03**/0.24	4/35.6/3.4	-

Figure 1: An 8×8 matrix partitioned naturally into partition 0 (top) and partition 1 (bottom). net-cut = 8

Figure 2: From Figure 1, move row 3 to partition 1, gain = 3. After the move net-cut = 5. In the gain column, a row is marked by "L" if it is locked.

Figure 3: From Figure 2, move row 5 to partition 0 with gain = 1. After the move net-cut = 4

Figure 4: From Figure 3, move row 2 to partition 1 with gain = 0. After the move net-cut = 4

Figure 5: From Figure 4, move row 7 to partition 0 with gain = 3. After the move net-cut = 1

Figure 6: From Figure 5, move row 1 to partition 1 with gain = -4. After the move net-cut = 5

Figure 7: From Figure 6, move row 8 to partition 0 with gain = -3. After the move net-cut = 8

Figure 8: From Figure 7, move row 4 to partition 1 with gain = 0. After the move net-cut = 8

Figure 9: From Figure 8, move row 6 to partition 0 with gain = 0. After the move net-cut = 8

Figure 10: Undo the steps back to Figure 5, After this net-cut = 1

Figure 11: From Figure 10, move the cut column (column 3) to the right-hand border, and reorder the other columns to give the bordered block diagonal form. The border is denoted by “B”.

Figure 12: multilevel algorithm applied to the west0381 test problem

$$\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ - \\ 5 \\ 6 \\ 7 \\ 8 \end{array} &
\left(\begin{array}{cccccccc}
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
 & \times & & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
 & \times & \times & \times & & \times & & \times \\
\times & & & & \times & & \times & \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times &
\end{array}\right) &
\begin{array}{c} 0 \\ 0 \\ 3 \\ 0 \\ - \\ 1 \\ 0 \\ 0 \\ 0 \end{array}
\end{array}$$

$$
\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 1 \\ 2 \\ 4 \\ - \\ 3 \\ 5 \\ 6 \\ 7 \\ 8 \end{array} &
\left(\begin{array}{cccccccc}
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
 & \times & & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
\times & & \times & & \times & & \times & \\
 & \times & \times & \times & & \times & & \times \\
\times & & & & \times & & \times & \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times &
\end{array}\right) &
\begin{array}{c} 0 \\ 0 \\ 0 \\ - \\ L \\ 1 \\ -3 \\ 0 \\ -3 \end{array}
\end{array}
$$

$$
\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 1 \\ 2 \\ 4 \\ 5 \\ - \\ 3 \\ 6 \\ 7 \\ 8 \end{array} &
\begin{pmatrix}
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times & \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times &
\end{pmatrix} &
\begin{array}{c} -1 \\ 0 \\ -1 \\ L \\ - \\ L \\ -3 \\ 3 \\ -3 \end{array}
\end{array}
$$

$$\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 1 \\ 4 \\ 5 \\ - \\ 2 \\ 3 \\ 6 \\ 7 \\ 8 \end{array} &
\left(\begin{array}{cccccccc}
 & \times & \times & \times & & \times & & \times \\
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times & \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times &
\end{array}\right) &
\begin{array}{c} -1 \\ -1 \\ L \\ - \\ L \\ L \\ -3 \\ 3 \\ -3 \end{array}
\end{array}$$

$$
\begin{array}{ccc}
 & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} & gain \\
\begin{matrix} 1 \\ 4 \\ 5 \\ 7 \\ - \\ 2 \\ 3 \\ 6 \\ 8 \end{matrix}
&
\begin{pmatrix}
 & \times & \times & \times & & \times & & \times \\
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times & \\
\times & & \times & & \times & & \times &
\end{pmatrix}
&
\begin{matrix} -4 \\ -4 \\ L \\ L \\ - \\ L \\ L \\ -3 \\ -3 \end{matrix}
\end{array}
$$

$$
\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 4 \\ 5 \\ 7 \\ - \\ 1 \\ 2 \\ 3 \\ 6 \\ 8 \end{array} &
\begin{pmatrix}
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times & \\
\times & & \times & & \times & & \times &
\end{pmatrix} &
\begin{array}{c} 0 \\ L \\ L \\ - \\ L \\ L \\ L \\ -3 \\ -3 \end{array}
\end{array}
$$

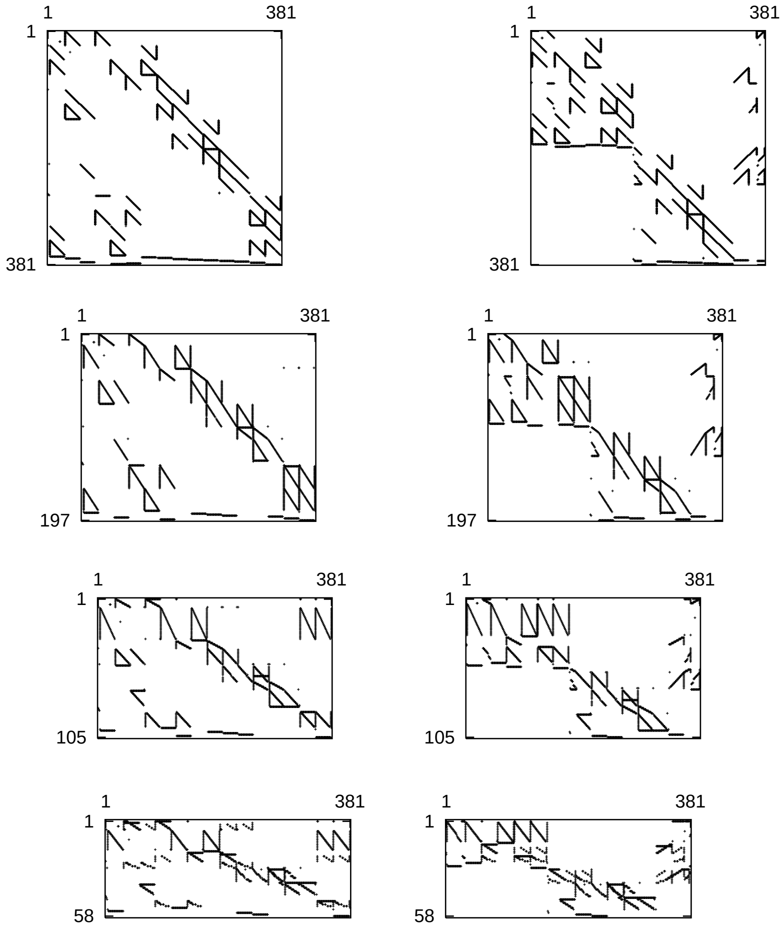
$$\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 4 \\ 5 \\ 7 \\ 8 \\ - \\ 1 \\ 2 \\ 3 \\ 6 \end{array} &
\begin{pmatrix}
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times & \\
- & - & - & - & - & - & - & - \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times &
\end{pmatrix} &
\begin{array}{c} 0 \\ L \\ L \\ L \\ - \\ L \\ L \\ L \\ 0 \end{array}
\end{array}$$

$$\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & gain \\
\begin{array}{c} 5 \\ 7 \\ 8 \\ - \\ 1 \\ 2 \\ 3 \\ 4 \\ 6 \end{array} &
\begin{pmatrix}
 & \times & \times & \times & & \times & & \times \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times & \\
- & - & - & - & - & - & - & - \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
 & \times & & \times & & \times & & \times \\
\times & & & & \times & & \times &
\end{pmatrix} &
\begin{array}{c} L \\ L \\ L \\ - \\ L \\ L \\ L \\ L \\ 0 \end{array}
\end{array}$$

$$
\begin{array}{ccc}
 & \begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{matrix} & gain \\
\begin{matrix} 5 \\ 6 \\ 7 \\ 8 \\ - \\ 1 \\ 2 \\ 3 \\ 4 \end{matrix} &
\begin{pmatrix}
 & \times & \times & \times & & \times & & \times \\
\times & & & & \times & & \times & \\
 & & \times & \times & & \times & & \times \\
\times & & \times & & \times & & \times & \\
- & - & - & - & - & - & - & - \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
 & \times & & \times & & \times & & \times
\end{pmatrix} &
\begin{matrix} L \\ L \\ L \\ L \\ - \\ L \\ L \\ L \\ L \end{matrix}
\end{array}
$$

$$
\begin{array}{ccc}
 & \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} & \mathit{gain} \\
\begin{array}{c} 1 \\ 4 \\ 5 \\ 7 \\ - \\ 2 \\ 3 \\ 6 \\ 8 \end{array}
&
\begin{pmatrix}
 & \times & \times & \times & & \times & & \times \\
 & \times & & \times & & \times & & \times \\
 & \times & \times & \times & & \times & & \times \\
 & & \times & \times & & \times & & \times \\
- & - & - & - & - & - & - & - \\
 & & \times & & & & & \\
\times & & \times & & \times & & \times & \\
\times & & & & \times & & \times & \\
\times & & \times & & \times & & \times &
\end{pmatrix}
&
\begin{array}{c} -4 \\ -4 \\ -4 \\ -3 \\ - \\ 0 \\ -3 \\ -3 \\ -3 \end{array}
\end{array}
$$

$$
\begin{array}{c}
 \\ 1 \\ 4 \\ 5 \\ 7 \\ - \\ 2 \\ 3 \\ 6 \\ 8
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} 2 & 4 & 6 & 8 & 1 & 5 & 7 & 3 \end{array} \\
\left(
\begin{array}{cccccccc}
\times & \times & \times & \times & & & & B \\
\times & \times & \times & \times & & & & \\
\times & \times & \times & \times & & & & B \\
 & \times & \times & \times & & & & B \\
- & - & - & - & - & - & - & - \\
 & & & & & & & B \\
 & & & & \times & \times & \times & B \\
 & & & & \times & \times & \times & \\
 & & & & \times & \times & \times & B
\end{array}
\right)
\end{array}
\begin{array}{c}
gain \\ -4 \\ -4 \\ -4 \\ -3 \\ - \\ 0 \\ -3 \\ -3 \\ -3
\end{array}
$$



1
381
1
381
1
381
1
381
1
381
1
197
1
381
1
197
1
381
1
105
1
381
1
105
1
381
1
58
1
381
1
58