DECOMPOSING LINEAR PROGRAMS FOR PARALLEL SOLUTION

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Ali Pinar
July, 1996

DECOMPOSING LINEAR PROGRAMS FOR PARALLEL SOLUTION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Ali Pınar

July, 1996

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Cevdet Aykanat(Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Mustafa Çelebi Pınar

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

[illegible]

Asst. Prof. Hakan Karaata

Approved for the Institute of Engineering and Science:

M. Baray

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

DECOMPOSING LINEAR PROGRAMS FOR PARALLEL SOLUTION

Ali Pınar
M. S. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Cevdet Aykanat
July, 1996

Many current research efforts are based on better exploitation of sparsity—common in most large scaled problems—for computational efficiency. This work proposes different methods for permuting sparse matrices to block angular form with specified number of equal sized blocks for efficient parallelism. The problem has applications in linear programming, where there is a lot of work on the solution of problems with existing block angular structure. However, these works depend on the existing block angular structure of the matrix, and hence suffer from unscalability. We propose two hypergraph models for decomposition, and these models reduce the problem to the well-known hypergraph partitioning problem. We also propose a graph model, which reduces the problem to the graph partitioning by node separator problem. We were able to decompose very large problems, the results are quite attractive both in terms solution quality and running times.

Key words: Sparse Matrices, Block Angular Form, Hypergraph Partitioning, Graph Partitioning by Node Separator

ÖZET

DOĞRUSAL PROGRAMLARIN PARALEL ÇÖZÜMLEME İÇİN BÖLÜNMESİ

Ali Pınar
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Danışman: Doç. Dr. Cevdet Aykanat
Temmuz, 1996

Birçok güncel araştırma büyük ölçekli problemlerin matrislerinde sıkça rastlanan seyreklikten daha iyi yararlanmaya dayalıdır. Bu araştırma, seyrek bir matrisi belli sayıda eşit büyüklükte bloklardan oluşan blok açısal duruma çevirmek için değişik metodlar önermektedir. Bu problemin önemli bir uygulaması doğrusal programlamadadır. Doğrusal programlamada, varolan blok açısal yapıları kullanan birçok çözüm yöntemi önerilmiştir. Ama bu yöntemler yalnızca varolan blok açısal duruma dayandıkları için ölçeklendirme sorunuyla karşı karşıyadırlar.

Bu çalışma bölünme için iki hiperçizge modeli öneriyor, ve bu modeller problemi iyi bilinen hiperçizge parçalama problemine indirgiyor. Önerilen bir diğer model ise çizge modeli, ve bu model de problemi düğüm ayıracıyla çizge parçalama problemine indirgiyor. Önerilen modeller, çok sayıda çok büyük ölçekli matrisleri bölmede denendi. Hem çözüm kalitesi, hem de zaman açısından çok çekici sonuçlar elde edildi.

Anahtar sözcükler: Seyrek Matris, Block Açısal Durum, Hiperçizge Parçalama, Düğüm Ayıracıyla Çizge Parçalama

To my family

Acknowledgment

I would like to express my deep gratitude to my supervisor Dr. Cevdet Aykanat for his guidance, suggestions, encouragement and enjoyable discussions throughout the development of the thesis. I would like to thank Dr. Mustafa Pınar for reading and commenting on the thesis. I would also like to thank Dr. Hakan Karaata for reading and commenting on the thesis. I owe special thanks to all members of the department for providing a pleasant environment for study. Finally, I am very grateful to my family and my friends for their support and patience.

Contents

List of Figures

List of Tables

1. Introduction

Studies on sparse matrices has its origins in diverse fields such as management science, power systems analysis, finite element problems, circuit theory, etc. Mathematical models in all of these areas give rise to very large systems of linear equations that could not be solved if most of the entries in these matrices were not zeros. This increases the interest in sparsity, because its exploitation can lead to enormous computational savings and because many large problems that occur in practice are sparse.

An important exploitation of sparsity arises in solving linear systems of equations. A good ordering of the rows and columns of the matrix can help us to preserve sparsity during factorization. The problem has been heavily studied in the literature because of the significant computational savings and the wide-applicability of the problem. Minimum Degree Ordering [25], Nested dissection [26] are the most popular solution methods of this problem. Other special forms of sparse matrices such as band matrices, block tridiagonal matrices, and block triangular matrices give rise to computational savings and special solution techniques, and permutation into these forms has been studied in the literature [20].

Although ordering sparse matrices to various special forms has been studied in the literature, the problem of permuting rows and columns of a sparse matrix into a block angular form, with specified number of equal sized blocks while minimizing the number of coupling rows, remains almost untouched. Solving linear systems of equations with block angular matrices has an inherent parallelism, because the blocks are independent, and can be handled concurrently. This kind of matrices arise in Linear Programming, such as multi-commodity flow, multi-stage stochastic problems.

Linear Programming (LP) is concerned with the optimization (maximization or minimization) of a linear function, while satisfying a set of linear equality and/or inequality constraints, and it is currently one of the most popular tools in modeling economic and physical phenomena where performance measures are to be optimized subject to certain requirements. LP was first conceived by George B. Dantzig around 1947. The most popular solution method for linear programming problems is the *Simplex Method* proposed by Dantzig in 1949. The other popular method is the *Interior Point Method* proposed by Karmarkar in 1984. Both of these methods have been successfully applied to many LP problems of moderate size. However, the performance of these two methods decreases as the problem size increases. The sizes of the constraint matrices of many LP's can be extremely large, in practice, which restricts the applicability of the standard solution techniques. This leads to the idea of applying divide-and-conquer schema for solving very large problems. Solving linear programs by decomposition was first proposed by Dantzig and Wolfe [18] in 1960, and has been the subject of many research efforts since then. Problems with block angular constraint matrices are very suitable for applying decomposition techniques. Also solution of these problems with decomposition has an inherent parallelism.

The parallel solution of block angular LP's has been a very active area of research in both operations research and computer science societies. The most popular decomposition technique, Dantzig–Wolfe decomposition has been successfully adopted for parallel solution of the block angular LP's. In this scheme, the block structure of the constraint matrix is exploited for parallel solution in the subproblem phase where each processor solves a smaller LP corresponding to a distinct block. A sequential coordination phase (the master) follows. This cycle is repeated until suitable termination criteria are satisfied. Coarse grain parallelism inherent in these approaches has been exploited in many other recent research works [27, 43]. However, the success of these approaches depends only on the existing *block angular* structure of the given constraint matrix. The number of processors utilized for parallelization in these studies is clearly limited by the number of inherent blocks of the constraint matrix. Hence, these approaches suffer from *unscalability* and *load imbalance*.

This work focuses on the problem of permuting rows and columns of an irregularly sparse rectangular matrix to obtain block angular structure with specified number of blocks for scalable parallelization. The objective in the decomposition is to minimize the number of coupling rows, while maintaining a balance criterion among the sizes of the blocks. Minimizing the number of coupling rows corresponds to minimizing the sequential component of the overall parallel scheme. Maintaining a balance criterion among the sizes of the blocks corresponds to minimizing processors' idle time during each subproblem phase.

The literature that addresses this problem is extremely rare and very recent. Ferris and Horn [21] model the constraint matrix as a bipartite graph. In this graph, each row and each column is represented by a vertex, and one set of vertices representing rows and the other set of vertices representing columns form a bipartition. There exists an edge between a row vertex and a column vertex if and only if the respective entry in the constraint matrix is nonzero. Ferris and Horn partition this graph using the Kernighan-Lin heuristic [36]. They obtain a node separator from this graph by repeatedly adding the vertex with highest degree to the separator. This enables permutation of the graph into a doubly bordered block angular form. Out of the vertices in the separator, ones representing the columns constitute the row-coupling columns, and ones representing the rows constitute the column-coupling rows. This doubly bordered matrix can be transformed into a block angular matrix by *column splitting*, a technique similar to the one used in stochastic programming to treat non-anticipativity [44]. This model naturally leads to a doubly bordered block angular matrix, and does not reduce the problem to any well-studied combinatorial optimization problem.

In this work, we propose three different models for representing sparse matrices for decomposition. Each model reduces the problem to a well-studied combinatorial optimization problem. In the first two models, we exploit hypergraphs to model matrices for decomposition. A hypergraph is defined as a set of vertices (nodes) and a set of *nets* (hyperedges) between those vertices. Each net is a subset of the vertices of the hypergraph. A graph is a special instance of a hypergraph, where each net contains exactly two vertices.

In the first model—referred to here as the *row-net* model—each row is represented by a net, whereas each column is represented by a vertex. The set of vertices connected to a net corresponds to the set of columns which have a nonzero entry in the row represented by this net [48]. In this case, the decomposition problem reduces to the well-known *hypergraph partitioning* problem which is known to be *NP-Hard* [24]. Hypergraph partitioning tries to minimize the number of nets on the cut, while maintaining balance between the parts. Maintaining balance corresponds to balance between sizes of the blocks in the block angular matrix, and minimizing the number of nets on the cut corresponds to minimizing the number of coupling rows in the block angular matrix.

The second model—referred to here as the *column-net* model—is very similar to the row-net model, only the roles of columns and rows are exchanged. Each column is represented by a net, whereas each row is represented by a vertex. The set of vertices connected to a net corresponds to the set of rows which have a nonzero entry in the column represented by this net [48]. Applying partitioning on this hypergraph can be considered as permuting the rows and columns of this matrix to dual block angular form. This dual block angular matrix achieved by hypergraph partitioning can be transformed into a block angular form by using column-splitting [44].

Hypergraph partitioning has been heavily studied in VLSI design automation, and many heuristics have been proposed for this problem. In this study, we make use of different heuristics originally proposed for VLSI partitioning, and adapt these heuristics for decomposing matrices.

In our third model—referred to here as the *Row Interaction Graph* model—each row is represented by a node, and there is an edge between two nodes if there exists a column which has nonzeros in both respective rows [47]. This model reduces the decomposition problem into the graph partitioning by node separator problem. Nodes in part P_i of a partition correspond to the rows in block B_i, and nodes in the separator correspond to the coupling rows. Hence, minimizing the number of nodes in the separator corresponds to minimizing the size of the master problem. By definition of the node separator, there are no edges between nodes in different parts, hence there is no interaction among rows of different blocks.

The problem of partitioning by node separators has applications in ordering matrices to preserve sparsity during factorization. Besides utilizing present methods for decomposition, this work includes contributions for finding better node separators on graphs.

We have demonstrate the validity of the proposed graph model with various linear program constraint matrices selected from *NETLIB* and other sources. We were able to decompose a matrix with 10099 rows, 11098 columns, 39554 nonzeros into 8 blocks with only 517 coupling rows in 1.9 seconds and a matrix with 34774 rows, 31728 columns, 165129 nonzeros into 8 blocks with only 1029 coupling rows in 10.1 seconds. The solution times with *LOQO* are 907.6 seconds for the former and 5970.3 seconds for the latter. These results are quite promising and our decomposition techniques form feasible decompositions for parallel solution.

The organization of this thesis is as follows: Chapter 2 includes a definition of block angular matrices and their applications. Chapter 3 presents a brief description of hypergraphs and the hypergraph partitioning problem. We use the terminology described in this section throughout the thesis. This chapter also reviews different approaches proposed for hypergraph partitioning problem such as local search methods, geometric embeddings, multi-level approaches and multi-start techniques. Chapter 4 defines the graph partitioning by node separators problem, and reviews the previous work. New methods that can help us to find better separators are also presented in this chapter. Chapter 5 describes the four models for permuting matrices to block angular form. We review the bipartite graph model of Ferris and Horn, and propose row-net, column-net, and row interaction graph models. Chapter 6 presents our experimental results, and comparisons of different models and methods, and comments on the experimental results. Finally, we give directions for future work and conclude the thesis in Chapter 7.

2. Block Angular Form of a Sparse Matrix

Block angular systems have been attractive for computer scientist due to their inherent parallelism. The blocks of the system can be handled concurrently, since they are independent. Below, we will define block angular systems, and we will discuss some of the applications.

2.1 Preliminaries

In this section, we will define block angular matrices. Definitions 2.1–2.4 have been taken from [21].

Definition 2.1 *A matrix $A \in \Re^{M\times N}$ is said to be in* block angular form *if it has the following structure:*

$$A = \begin{pmatrix} B_1 & & & \\ & B_2 & & \\ & & & \\ & & & B_k \\ R_1 & R_2 & \dots & R_k \end{pmatrix}$$

where $B_i \in \Re^{m_i \times n_i}, R_i \in \Re^{q\times n_i}$. Each submatrix B_i is called a block, and

$$M = \sum_{i=1}^{k} m_i + q \text{ and } N = \sum_{i=1}^{k} n_i \ .$$

Definition 2.2 *A matrix $A \in \Re^{M\times N}$ is said to be in* dual block angular form *if it has the following structure:*

$$A_B^d = \begin{pmatrix} B_1 & & & C_1 \\ & B_2 & & C_2 \\ & & & \vdots \\ & & B_k & C_k \end{pmatrix}$$

where $B_i \in \Re^{m_i \times n_i}$, $, C_i \in \Re^{m_i \times p}$. *Each submatrix* B_i *is called a block, and*

$$M = \sum_{i=1}^{k} m_i \text{ and } N = \sum_{i=1}^{k} n_i + p \ .$$

Definition 2.3 *A matrix* $A \in \Re^{M \times N}$ *is said to be in* doubly bordered block angular form *if it has the following structure:*

$$A_{DB} = \begin{pmatrix} B_1 & & & C_1 \\ & B_2 & & C_2 \\ & & & \vdots \\ & & B_k & C_k \\ R_1 & R_2 & R_k & D \end{pmatrix}$$

where $B_i \in \Re^{m_i \times n_i}$, $C_i \in \Re^{m_i \times p}$, $R_i \in \Re^{q \times n_i}$ *and* $D \in \Re^{q \times p}$. *Each submatrix* B_i *is called a block, and*

$$M = \sum_{i=1}^{k} m_i + q \text{ and } N = \sum_{i=1}^{k} n_i + p \ .$$

Definition 2.4 *Each row of the* $q \times N$ *submatrix*

$$(R_1 \ R_2 \ \ldots \ R_k \ D)$$

is called a column-linking or column-coupling row.

Generally, column-linking rows restrict the column spaces of the blocks, resulting in the column space for the entire matrix. A column-linking row may restrict the column space of one block B_i based on the column space of another block B_j. In this case, the blocks B_i and B_j are said to be *linked* or *coupled* by this row.

Definition 2.5 *Each column of the* $M \times p$ *submatrix*

$$\begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_k \\ D \end{pmatrix}$$

is called a row-linking or row-coupling column.

Generally, row-linking columns restrict the row spaces of the blocks, resulting in the row space for the entire matrix. A row-linking column may restrict the row space of one block B_i based on the row space of another block B_j. In this case, the blocks B_i and B_j are said to be *linked* or *coupled* by this column.

2.2 Block Angular Systems in Linear Programming

Linear Programming (LP) is concerned with the optimization (maximization or minimization) of a linear function, while satisfying a set of linear equality and/or inequality constraints. The linear programming problem was first conceived by George B. Dantzig around 1947.

A linear program has the canonical form

$$\begin{array}{lllllll}
\textit{Minimize} & c_1x_1 & + \; c_2x_2 & + \; \ldots & + \; c_nx_n & & \\
\textit{Subject to} & & & & & & \\
& a_{11}x_1 & + \; a_{12}x_2 & + \; \ldots & + \; a_{1n}x_n & \geq & b_1 \\
& a_{21}x_1 & + \; a_{22}x_2 & + \; \ldots & + \; a_{2n}x_n & \geq & b_2 \\
& \vdots & \vdots & \ddots & \vdots & & \vdots \\
& a_{m1}x_1 & + \; a_{m2}x_2 & + \; \ldots & + \; a_{mn}x_n & \geq & b_m \\
& x_1 & , \; x_2 & , & , \; x_n & \geq & 0
\end{array}$$

Here $c_1x_1 \; + \; c_2x_2 \; + \; \ldots \; + \; c_nx_n$ is the *objective function*, $c_1, c_2, \ldots, c_n$ are *objective coefficients*, and $x_1, x_2, \ldots, x_n$ are *decision variables* to be determined. The inequality $\sum_{j=1}^{n} a_{ij}x_j \geq b_i$ denotes the ith constraint. The coefficients a_{ij} for $i = 1, 2, \ldots, m$, and $j = 1, 2, \ldots, n$ are called the *technological coefficients*, and they form the *constraint matrix* A.

$$A = \begin{pmatrix} a_{11} & a_{12} & & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}$$

The column vector whose ith component is b_i, which is referred to as the *right-hand side vector* represents the minimal requirements to be satisfied. Hence, using this matrix notation an LP problem can be represented as:

$$\begin{aligned} \mathit{Minimize} \quad & c^T x \\ \mathit{Subject\ to} \quad & A^T x \geq b \\ & x \geq 0 \end{aligned}$$

Every LP problem has an associated *dual problem.* The dual problem for the foregoing problem can be stated as :

$$\begin{aligned} \mathit{Maximize} \quad & b^T y \\ \mathit{Subject\ to} \quad & A^T y \leq c \\ & y \geq 0 \end{aligned}$$

A set of variables $x_1, x_2, \ldots, x_n$ satisfying all the constraints is called a *feasible point.* The set of all such points constitutes the *feasible region.* Using these definitions, the linear programming problem can be stated as follows: *Among all feasible points, find one that minimizes (or maximizes) the objective function [4].*

The most popular solution method for linear programming problems is the "Simplex Algorithm", which was proposed by Dantzig in 1949. It has been widely accepted for its simplicity to understand and implement, and its speed in small sized problems. It is a local search algorithm, and moves from one extreme point to another, and it finds the optimal solution, since the set of feasible points is a convex set. The asymptotic complexity of the algorithm is exponential in the worst-case.

Another popular method for solving linear programming problems is the *interior point* methods, which started with the pioneering work of Karmarkar in 1984 [34]. As the name implies this method moves in the inner space of the feasible region, and finds an optimal solution. The important point in Karmarkar's method is its polynomial asymptotic complexity.

The performance of both methods decreases as the problem size increases. Also memory becomes restrictive for large problems. This leads to the idea of solving linear programs by decomposition. The first decomposition scheme was proposed by Dantzig and Wolfe in 1960 [18]. In this scheme, the problem is decomposed into subproblems. Each time a subproblem is solved, and the results are used in the solution of the forecoming subproblems.

If the systems is block angular, each block corresponds to a subproblem, and these subproblems can be solved concurrently, since they are independent. Multi-commodity flow, multi-item production scheduling, economic development problems and multi-stage stochastic problems has block angular constraint matrices [7].

Starting with the pioneering work of Dantzig and Wolfe in 1960 [18], solution of block angular problems (either in parallel or in serial) has been an active area of study, and lead to several studies. Bender decomposition [5], Bundle-based decomposition [43], Alternating Directions Method [38] are examples of such work.

However, the literature addressing how to obtain a block angular structure of a sparse matrix is very rare and recent. Solution methods for this problem will be discussed in Chapter 5.

3. Graph and Hypergraph Partitioning

The importance and popularity of the graph partitioning problem is mostly due to its connection to the problems whose solution depend on the divide-and-conquer paradigm. A partitioning algorithm partitions a problem into semi-independent subproblems, and tries to reduce the interaction between these subproblems. This division of a problem into simpler subproblems results in a substantial reduction in the search space. Graph Partitioning is the basis of hypergraph partitioning, which is more general and more difficult. Graph partitioning has a number of important applications. An exhaustive list of these applications combined with the relevant references is given below.

- VLSI placement [41]
- VLSI routing [57]
- VLSI circuit simulation [1]
- memory segmentation to minimize paging [36]
- mapping of tasks to processors to minimize communication [10]
- efficient sparse Gaussian elimination [26]
- laying out of machines in advanced manufacturing systems [56]

Some applications of the hypergraph partitioning problem are listed below:

- VLSI placement [22]
- VLSI routing [53]

We can extend the adjacency definition of a vertex to adjacency definition of a set of vertices $V \subseteq \mathcal{V}$ as follows:

$$Adj(V) \equiv \bigcup_{v \in V} Adj(v) - V \ .$$

We use $Adj(v, U)$ to denote the set of vertices adjacent to v in U.

$$Adj(v, U) \equiv Adj(v) \cap U \ .$$

Extending this definition to sets, we can say

$$Adj(V, U) \equiv Adj(V) \cap U \ .$$

We will use $Adj_{\mathcal{E}}(v)$ to denote the set of edges adjacent to vertex v.

$$Adj_{\mathcal{E}}(v) \equiv \{e | e \in \mathcal{E} \ and \ (e = (u, v) \ or \ e = (v, u))\} \ .$$

For hypergraphs $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ (also for graphs), a weight function can be defined to map each vertex to a positive number. A similar function can be defined to map nets to positive numbers. We will call the former function the *weight* function, the latter function as the *cost* function. We can extend the definition of *cost* and *weight* functions for sets as follows:

$$weight(\mathcal{V}') = \sum_{v \in \mathcal{V}'} weight(v) \ \ for \ all \ \ \mathcal{V}' \subseteq \mathcal{V}$$

$$cost(\mathcal{N}') = \sum_{n \in \mathcal{V}'} cost(n) \ \ for \ all \ \mathcal{N}' \subseteq \mathcal{N}$$

Below we will discuss the definition of partitioning for hypergraphs.

Definition 3.1 *$P = \{P_1, P_2, \ldots, P_k\}$ is a k-way partition of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ if and only if the following three conditions hold:*

- $P_i \subset \mathcal{V}$ *and* $P_i \neq \emptyset$ *for* $1 \leq i \leq k$
- $\bigcup_{i=1}^{k} P_i = \mathcal{V}$
- $P_i \cap P_j = \emptyset$ *for* $1 \leq i < j \leq k$

When $k = 2$ this partition is called as a *bisection* or a *bipartition.*

For a partition P, a net n is said to be internal in partition P_i, if and only if

$$\forall v \in n,\ v \in P_i \qquad \text{or} \qquad n \cap P_i = n$$

The set of internal nets $\mathcal{N}_I$ is defined as

$$\mathcal{N}_I = \{n | n \ is\ an\ internal\ net\ in\ a\ part\}$$

or

$$\mathcal{N}_I = \{n | n \cap P_i = n\ \ for\ \ n \in \mathcal{N}\ \ and\ \ P_i \in P\}$$

and the set of external nets $\mathcal{N}_E$ is defined as

$$\mathcal{N}_E = \{n | n \cap P_i \neq \emptyset\ \ and\ \ n \cap P_i \neq n\ \ for\ \ n \in \mathcal{N}\ \ and\ \ P_i \in P\}.$$

There are different functions for the cost of a cut. Two of these are widely used. The first one is the number of nets on the cut. In this metric cutsize $\mathcal{C}(P)$ can be defined as:

$$\mathcal{C}(P) = \sum_{n \in \mathcal{N}_E} cost(n) = cost(\mathcal{N}_E) = cost(\mathcal{N}) - cost(\mathcal{N}_I)$$

The second metric is the *connectivity* metric. The connectivity of a net is equal to the number of parts it connects. Formally, connectivity of net $con(n)$ is

$$con(n) = |\{P_i : 1 \leq i \leq k\ \ and\ \ P_i \cap n \neq \emptyset\}|$$

With this metric, cutsize $\mathcal{C}(P)$ is defined as:

$$\mathcal{C}(P) = \sum_{n \in \mathcal{N}} con(n)$$

A partition is *balanced* if all parts have about the same weight. When all parts have exactly the same weight, we call this partitioning as *perfectly balanced.* A formal definition for the balance criterion can be expressed as :

$$\frac{W_{max} - W_{avg}}{W_{max}} \leq \epsilon$$

where W_{max} is the weight of the part with maximum weight, W_{avg} is the average weight of parts (i.e., $W_{avg} \frac{\sum_{v \in \mathcal{V}} weight(v)}{k}$), and ϵ is a predetermined imbalance ratio.

In the light of the definitions above we can define the hypergraph partitioning problem as finding a balanced partition P which minimizes the cost function $\mathcal{C}(P)$.

3.2 Local Search Heuristics

Local search heuristics are very popular for solving combinatorial optimization problems, since they can be easily implemented and they can be very fast. A general overview of a local search heuristic is given in Figure 3.1. A local search heuristic starts with a random feasible solution, then iteratively improves this solution by moving to solutions in the neighborhood space. This leads to two critical points for local search methods: definition of a neighborhood space and how to choose the neighbor to move within this space. This section discusses neighborhood structures and various methods proposed for the solution of the graph and hypergraph partitioning problems.

```
Input : A combinatorial optimization problem
Output : A local optimum solution

1.   generate an initial feasible solution $s$ for the problem
2.   repeat
2.1      Find a neighbor $s'$ of $s$ with $cost(s) \geq cost(s')$
2.2      $s \leftarrow s'$
     until no improvement on $s$ is possible.
```

Figure 3.1. A general view of a local search heuristic

3.2.1 Neighborhood Structures for the Hypergraph Partitioning Problem

The most critical part of a local search heuristic is the neighborhood definition. After the definition of the neighborhood structure second critical choice is how to choose a neighbor s' for a solution s from the neighborhood $N(s)$. Generally there are three ways:

- *First descent method* chooses the first neighbor in $N(s)$ that has a better cost than s.

- *Steepest descent method* chooses the neighbor $s\prime$ which gives the best cost among all solutions in $N(s)$.
- *Random descent method* chooses a neighbor randomly among solutions in $N(s)$.

First descent method is faster than the steepest descent method, however steepest descent has a higher chance to produce better results.

There are two popular neighborhood structures for the graph partitioning problem. The first is the *Swap-Neighborhood* and the second is the *Move-Neighborhood*. Below, we will discuss these two neighborhood structures.

- **Swap–Neighborhood:**

  In this neighborhood structure, two partitions are neighbors if one partition can be obtained from another by swapping two vertices between different parts in one of the partitions. Formally,

  Definition 3.2 *Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a n-vertex hypergraph and P, P' two k-way partitions of $\mathcal{H}$. Then, the partition $P = (P_1, \ldots, P_i, \ldots, P_j, \ldots, P_k)$ and the partition $P' = (P_1, \ldots, (P_i - \{v\}) \cup \{u\}, \ldots, (P_j - \{u\}) \cup \{v\}, \ldots, P_k)$ are neighbors for some vertices $v \in P_i$, $u \in P_j$.*

  The partition P has $(k(k-1)/2)(n/k)^2$ neighbors if each part has n/k vertices.

- **Move–Neighborhood:**

  A partition $P = (P_1, \ldots, P_k, \ldots, P_l, \ldots, P_k)$ has a move-neighbor partition P' if P' can be obtained from P by moving a vertex from one part to another in P. Formally,

  Definition 3.3 *Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a n-vertex graph and P, P' two k-way partitions of $\mathcal{H}$. Then, the partition $P' = (P_1, \ldots, P_i - \{v\}, \ldots, P_j \cup \{v\}, \ldots, P_k)$ is a move-neighbor of the partition $P = (P_1, \ldots, P_i, \ldots, P_j, \ldots, P_k)$ for some $P_i, P_j \in P$ and for some vertex $v \in P_i$.*

  The partition P has at most $k(k-1)(n/k) = n(k-1)$ neighbors if each part has n/k vertices.

The Swap-Neighborhood space of a partition is larger than its Move-Neighborhood, which enables a better search in neighborhood of a solution. However, searching the swap-neighborhood takes more time compared to searching the move-neighborhood. Different algorithms using these two neighborhood spaces will be described in Section 3.2.4.

3.2.2 Hill-climbing

Local search heuristics can be supported with a hill-climbing feature. Suppose the problem is a minimization problem, and $cost(s)$ denotes the cost of the solution s. If we choose $s\prime$ with $cost(s\prime) < cost(s)$, this will be a *downhill step.* On the other hand, if we choose $s\prime$ with $cost(s\prime) > cost(s)$, this will be an *uphill step.* Allowing uphill moves enables the heuristic to escape from being trapped in a local optima. In hypergraph partitioning, generally all possible moves (swaps) are examined and only a prefix of the moves (a consecutive subset of the moves starting with the first move) giving the best solution are realized. This enables us to escape from local optimas and find better partitions. A local search heuristic with hill-climbing is given in Figure 3.2.

3.2.3 Tie-breaking Strategies

A critical decision in the iterative improvement methods is the choice of the vertex to move (vertex pair to swap). Most of the time, there is more than one vertex, which gives the same improvement. Hagen et. al. [28], observe that 15 to 30 vertices typically share the highest gain value at any time during an FM pass on a VLSI circuit with 833 modules (Primary1). This shows that an intelligent tie-breaking mechanism can make significant improvements on the overall performance of the partitioning algorithm.

One simple decision is to use LIFO, FIFO or random strategy to choose the vertex to move, out of the highest gain moves. Hagen et. al., experimented with these three strategies and showed that LIFO strategy significantly overperforms the other two [28]. One explanation for this success will be that LIFO enforces "locality" for choosing vertices to move. That is, vertices that form a natural cluster will probably move sequentially.

Input: A combinatorial optimization problem
Output: A local optimum solution

```
1       generate an initial feasible solution $s$ for the problem
2       repeat
2.1         for a limited number of iterations do
2.1.1           select the best neighbor $s\prime$ of in $N(s)$.
2.1.2           $s \leftarrow s'$
2.3     find the prefix of steps from the loop above, which leads
            to the best solution in this pass.
2.4         if the prefix is non-empty, then
2.4.1           realize the steps in this prefix.
2.5         else a local optimum has been found.
        until a local optimum has been found.
```

Figure 3.2. A generalized local search algorithm with hill-climbing

Another method for tie-breaking is to consider not only the gain for that move, but also some possible gains for the future moves. Krishnamurthy introduced a *gain vector*, which is a sequence of potential gain values corresponding to gains of possible moves in the future. The *rth* entry in the gain vector considers the gain r moves ahead. Ties are broken by first considering first level gain, then the second level gain, etc.

Hagen et. al.[28], introduces a similar *look-ahead* ability by improving Krishnamurthy's gain computation with the basic idea behind the success of LIFO structure.

All these studies show the importance of tie-breaking strategies for iterative improvement methods. This is still a hot topic for hypergraph partitioning problem.

3.2.4 Hypergraph Partitioning Heuristics

In this section, we will overview different hypergraph partitioning heuristics, which use a local search strategy.

3.2.4.1 Kernighan-Lin's Approach

Kernighan-Lin (KL) heuristic was originally proposed for the graph partitioning problem. This heuristic is a local search algorithm and has become the basis of many graph and hypergraph partitioning heuristics [36]. KL heuristic uses a Swap-Neighborhood structure (described in Section 3.2.1). In this neighborhood structure, two partitions are neighbors if one partition can be obtained from another by swapping two vertices between different parts in one of the partitions.

This heuristic assumes that every vertex has the same weight. It works as follows: first, an initial partition is generated. We then determine the vertex pair whose swap results in the largest swap gain, i.e., the largest decrease in the cutsize or the smallest increase (if no decrease is possible). This pair is tentatively interchanged and locked. The locking prohibits them from taking part in future swaps in this pass. Then, we look for a second pair of vertices whose interchange improves the cutsize the most, and do the same for this pair also. We continue in this way, but we keep a record of all tentative swaps and their gains. We finish when all the vertices are locked. At this time, we have interchanged both parts and are back to the original (initial) cutsize. Starting with the first swap in the record, we perform the subsequence of swaps which result in the smallest cutsize. The following pass begins with unlocking all vertices and proceeds in the same manner. These passes are repeated until there is no improvement in the cutsize which corresponds to a locally minimum partition.

KL heuristic allows uphill moves to reduce the danger of being trapped in a poor local minimum. This feature of the heuristic enables it to produce better partitions than the heuristics that employ only downhill moves. Also, this algorithm is quite robust. We can accommodate additional constraints such as partitioning into unequal-sized parts, required parts for certain vertices. However, it has some disadvantages. The algorithm handles only identical vertex weights. This restricts the applicability of this heuristic. The algorithm has a complexity of $O(n^2 \lg n)$ per pass for a graph with n vertices. It has been observed that the algorithm performs poorly on sparse graphs. Furthermore, the quality of the solution generated by this heuristic strongly depends on the initial partition, just like any other local search partitioning method.

This heuristic has been initially proposed for graphs. The first studies on hypergraph partitioning with this heuristic was done by transforming the hypergraph to a graph. Later, Schweikert and Kernighan improved this method to handle hypergraphs [55]. A recent study by Dutt decreased the worst-case complexity of the KL algorithm to $\theta(max(|\mathcal{E}|d, |\mathcal{E}| \lg |\mathcal{V}|))$ and average complexity to $\theta(|\mathcal{E}| \lg |\mathcal{V}|)$, where d is the maximum vertex degree in $\mathcal{G}$.

3.2.4.2 Fiduccia-Mattheyses' Approach

Fiduccia-Mattheyses (FM) heuristic [22] was originally proposed for the hypergraph partitioning problem, but it can be applied to the graph partitioning problem as well. This algorithm introduces the Move-Neighborhood structure instead of the swap-neighborhood structure.

In addition, an efficient data structure called the bucket list data structure is proposed. This data structure helps to sort the vertices with respect to their move gains, in time linear in the number of the vertices and keep the vertices in a sorted order according to their move gains, during the partitioning iterations. Moreover, it reduces the time complexity of the KL heuristic to $O(|V| + |E|)$. These features of FM heuristic, made it the basis for many of the heuristics that followed.

3.2.4.3 Krishnamurthy's Approach

This heuristic [39] is an extension of FM's method. Look-ahead ability is added to the cell gain concept by considering the number of pins of a net in a part. Each node has a gain vector with size l, where l is the number of levels. First level gain is the same as that in FM's method. Second level gain shows the possible cutsize reduction in the next move which follows the current cell move. If a net has 2 cells in part P_1, and at least one cell in other part P_2, moving one of the two cells from P_1 to P_2 does not reduce the cut. Therefore effect of this net on first level gain of those cells are 0 and effect on the second level gains is the cost of this net.

3.2.4.4 Sanchis' Approach

Sanchis [54] generalized Krishnamurthy's algorithm to a multi-way hypergraph partitioning algorithm so that it could directly handle the partitioning of a hypergraph into more than two parts. All the previous approaches before Sanchis' algorithm (SN algorithm) are originally bipartitioning algorithms. Level 1 SN algorithm is briefly described here for the sake of simplicity of presentation. Details of SN algorithm which adopts multi-level gain concept can be found in [54]. In SN algorithm, each vertex of the hypergraph is associated with $(k-1)$ possible moves. Each move is associated with a *gain*. The *move gain* of a vertex v_i in part s with respect to part t $(t \neq s)$, i.e., the gain of the move of v_i from the home (source) part s to the destination part t, denotes the amount of decrease in the number of cut nets (cutsize) to be obtained by making that move. Positive gain refers to a decrease, whereas negative gain refers to an increase in the cutsize. Figure 3.3 illustrates the pseudo-code of the SN based k-way hypergraph partitioning heuristic. In this figure, $nets(v)$ denotes the set of nets incident to vertex v. The algorithm starts from a randomly chosen feasible partition (Step 1), and iterates a number of passes over the vertices of the hypergraph until a locally optimum partition is found (*repeat-loop* at Step 2). At the beginning of each pass, all vertices are *unlocked* (Step 2.1), and initial $k-1$ move gains for each vertex are computed (Step 2.2). At each iteration (*while-loop* at Step 2.4) in a pass, a feasible move with the maximum *gain* is selected, tentatively performed, and the vertex associated with the move is *locked* (Steps 2.4.1–2.4.6). The locking mechanism enforces each vertex to be moved at most *once* per pass. That is, a locked vertex is not selected any more for a move until the end of the pass. After the move, the move gains affected by the selected move should be updated so that they indicate the effect of the move correctly. Move gains of only those unlocked vertices which share nets with the vertex moved should be updated. Gain re-computation scheme is given here instead of gain update mechanism for the sake of simplicity in the presentation (Step 2.4.7). At the end of each pass, we have a sequence of tentative vertex moves and their respective gains. We then construct from this sequence the *maximum prefix subsequence* of moves with the *maximum prefix sum* (Steps 2.5 and 2.6). That is, the gains of the moves in the maximum prefix subsequence give the maximum decrease in the cutsize among all prefix subsequences of the moves tentatively performed. Then, we permanently realize the moves in the maximum prefix subsequence and start the next pass if the maximum prefix sum is positive. The partitioning process terminates if the maximum prefix sum is not positive, i.e., no further decrease in the cutsize is possible, and we then have found a locally optimum partitioning. Note that moves with negative gains, i.e., moves which increase the cutsize, might be selected during the iterations in a pass. These moves are tentatively realized in the hope that they will lead to moves with positive gains in the following iterations. This feature together with the maximum prefix subsequence selection brings the *hill–climbing* capability to the KL-based algorithms.

Figure 3.4 illustrates the pseudo-code of the move gain computation algorithm for a vertex u in the hypergraph. In this algorithm, $part(v)$ for a vertex $v \in \mathcal{V}$ denotes the part which the vertex belongs to, and $\sigma_n(t)$ counts the number of pins of net n in part t. Move of vertex u from part s to part t will decrease the cutsize if and only if one or more nets become internal net(s) of part t by moving vertex u to part t. Therefore, all other pins ($|n|-1$ pins) of net n should be in part t. This check is done in Step 3.3.1.

In this heuristic, each part contains $k-1$ bucket lists; one for each other part, a vertex can move. The time complexity of one pass is $O(l.p.k(\lg k + G_{max}.l))$, where l is the number of levels and G_{max} is the size of buckets.

3.2.4.5 Simulated Annealing

Simulated Annealing (SA) was proposed by Kirkpatrick [37] as an optimization method, which has the capability to escape from local minima. To solve a combinatorial optimization problem with SA, a neighborhood structure should be defined for the solution space. Then SA starts with an initial solution, picks a random neighbor of the current solution and moves to this neighbor if it represents a downhill move. Even if the new solution represents an uphill move, SA will move to it with probability $e^{\frac{-\delta}{T}}$, and stay in the current solution otherwise. Here δ is the improvement in the cost function of the problem, and T is the current value of the *Temperature parameter*. To control the rate of convergence and the way of searching the solution space, typically *temperature schedule* is

```
1       construct a random, initial, feasible partition;
2       repeat
2.1       unlock all vertices;
2.2       compute $k-1$ move gains of each vertex $v \in V$
            by invoking $computegain(H, v)$;
2.3       $mcnt = 0$;
2.4       while there exists a feasible move of an unlocked vertex do
2.4.1       select a feasible move with max gain $g_{max}$ of an unlocked vertex $v$
              from part $s$ to part $t$;
2.4.2       $mcnt = mcnt + 1$;
2.4.3       $G[mcnt] = g_{max}$;
2.4.4       $Moves[mcnt] = \{v, s, t\}$;
2.4.5       tentatively realize the move of vertex $v$;
2.4.6       lock vertex $v$;
2.4.7       update the move gains of unlocked vertices $u \in nets(v)$
              by invoking $computegain(H, u)$;
2.5       perform prefix sum on the array $G[1 \ldots mcnt]$;
2.6       select $i^*$ such that $G_{max} = \max_{1 \leq i^* \leq mcnt} G[i^*]$;
2.7       if $G_{max} > 0$ then
2.7.1       permanently realize the moves in $Moves[1 \ldots i^*]$;
        until $G_{max} \leq 0$;
```

Figure 3.3. Level 1 SN hypergraph partitioning heuristic

established, which modifies T as a function of current status of the search process (e.g., the number of moves). It has been shown that SA will converge to a globally optimum solution given an infinite number of moves and a temperature schedule that *cools* to zero sufficiently slowly. The terms "cooling" and "temperature schedule" are due to SA's analogy to physical annealing of a material into a ground-state energy configuration. One problem with SA is, at low temperatures, many candidate moves might be generated and rejected before one is finally accepted, significantly increasing the run-time.

SA overperforms, in the quality of cutsize, all the previous explained KL-based approaches. However, its run-time is too large, which makes it impractical.

```
computegain(H, u)
1           $s \leftarrow part(u)$;
2           for each part $t \neq s$ do
2.1             $g_u(t) \leftarrow 0$;
3           for each net $n \in nets(u)$ do
3.1             for each part $t = 1, \ldots, k$ do
3.1.1               $\sigma_n(t) \leftarrow 0$;
3.2             for each vertex $v \in n$ do
3.2.1               $p \leftarrow part(v)$;
3.2.2               $\sigma_n(p) \leftarrow \sigma_n(p) + 1$;
3.3             for each part $t \neq s$ do
3.3.1               if $\sigma_n(t) = |n| - 1$ then
3.3.1.1                 $g_u(t) \leftarrow g_u(t) + 1$;
```

Figure 3.4. Gain computation for a vertex u

3.2.4.6 Mean Field Annealing

Mean Field Annealing (MFA) is a technique similar to SA which also has a physical analogy to systems of particles in thermal equilibrium. It was first applied to graph partitioning by Van den Bout and Miller [19]. They use an indicator vector of size $|\mathcal{V}|$ to denote a bipartitioning solution, where $x_i = 0$ corresponds to placing v_i in the first part, and $x_i = 1$ corresponds to placing v_i in the second part. However, the value of x_i varies between 0 and 1. Initially, each x_i is set to a value slightly greater than 0.5. Next, a random vertex v_i is selected iteratively, and two solutions $\vec{x}(0)$ which places x_i in the first part, and $\vec{x}(1)$ which places x_i in the second part are considered. Then the value of x_i is updated as:

$$x_i = (1 + e^{(F(\vec{x}(1)) - F(\vec{x}(0)))/T})^{-1}$$

The intuition behind this calculation is x_i approaches its natural value after each update (i.e., x_i approaches to 1 iff $F(\vec{x}(1)) << F(\vec{x}(0))$, and to 0 iff $F(\vec{x}(0)) << F(\vec{x}(1))$).

The process of computing a new x_i for randomly chosen v_i is repeated until a stable solution is reached. The temperature T is then lowered and the process is repeated, moving x_i values further to 0 or 1. Finally, a graph bipartitioning solution can be obtained by rounding each x_i to its nearest discrete value. Bultan and Aykanat have extended this basic approach to multi-way partitioning of hypergraphs [9, 10].

The quality of MFA solutions are competitive with those of SA, but usually takes less time than SA, but MFA is still slower than KL-based approaches.

3.2.5 Alternative Strategies

A possible weakness of the KL and FM strategies lies in the locking mechanism, e.g., a module v may be moved from P_1 to P_2 early in a pass, but its neighbors in P_2 start moving to P_1, which causes v to be in the wrong cluster. To prevent such cases, Hoffman [32] proposed a dynamic locking mechanism which behaves like FM, except that when v is moved out of P_i every module in $Adj(v, P_i)$ becomes unlocked. This allows the neighbors of v in P_i to also migrate out of P_i. The algorithm permits a maximum of ten moves per module per pass. Daşdan and Aykanat proposed a multi-way FM variant that allows a small constant number vertex moves per pass [17].

3.2.6 Multi-start Techniques

As discussed in previous sections, iterative improvement heuristics are usually very fast, however they have a high tendency to be trapped in local minima. On the other hand, Simulated Annealing (SA) is very slow, but it is guaranteed to find a global optimum (given infinite time), and in practice SA overperforms other heuristics in quality of the solution at the expense of large amounts of CPU time.

One alternative to SA is a multi-start technique, running iterative improvement heuristics several times, each time starting from a different initial solution, and return the best result. This technique makes a significant improvement on the performance of iterative methods. Also, multi-start technique has a trivial parallelism, which gives an important advantage if the problem is a preprocessing step for a parallel application.

Nevertheless, the performance of multi-start technique becomes limited as the problem size increases. Boese et. al. [6], proposed an adaptive multi-start technique. The technique depends on using the knowledge obtained from previous solutions. One way is to group vertices, that have been placed in the same part, on previous runs for the initial partition. An alternative way is to form clusters of vertices that have been placed in the same part in all previous runs. Studies with adaptive multi-start techniques report improvements compared to pure multi-start techniques, especially for the average case performance.

3.3 Geometric Embeddings

A geometric representation of the hypergraph can provide a useful basis for a partitioning heuristic, since speedups and special "geometric" heuristics become possible. For example computing a minimum spanning tree of a weighted undirected graph requires $O(n^2)$ time, but the complexity reduces to $O(n \lg n)$ for points in a 2-dimensional space [51]. In this section, we will discuss partitioning graphs with the help of embedding the vertices into geometric space [3][12][23]. The three important representations are :

- **One-dimensional Representation:**

  One dimensional representation is a sequential list of the vertices. Generally, modules that are closely connected should lie close to each other in the ordering, so that the ordering can reveal the structure of the hypergraph.

- **Multi-dimensional Representation:**

  Multi-dimensional Representation is a set of n points in d-dimensional space with $d > 1$, where each point represents a single vertex. This representation implicitly defines a distance relation between every pair of modules. Geometric clustering algorithms can be applied to these points for the partitioning solution.

- **Multi-dimensional Vector Space Representation:**

  Using the multi dimensional vector space model, the vector space consists of indicator n-vectors (corresponding to bipartitioning solutions), and the problem becomes finding the direction of the best indicator vector.

Spectral methods have crucial importance for embedding graphs into geometric space. Assume that the hypergraph is represented as a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $A(a_{ij})$ (e.g., by replacing each net by a clique). The $n \times n$ degree matrix $D(d_{ij})$ is given by $d_{ii} = deg(v_i)$ and $d_{ij} = 0$ if $i \neq j$. The $n \times n$ *Laplacian* matrix of $\mathcal{G}$ is defined as $Q = D - A$. An n-dimensional vector $\vec{\mu}$ is an *eigenvector* of Q with *eigenvalue* λ if and only if $Q\vec{\mu} = \lambda\vec{\mu}$. We denote the set of eigenvectors of Q by $\vec{\mu_1}, \vec{\mu_2}, \ldots, \vec{\mu_n}$ with eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$.

The eigenvectors can be used to embed vertices into geometric space. Hall [29] uses the second small eigenvector μ_2 to order vertices. He simply sorts the values in the eigenvector μ_2, then divides the vertices into two with respect to this order. Vertices can be mapped to n-dimensional space, by using the other eigenvectors, other geometric clustering methods can be applied on these points. The survey paper by Alpert and Kahng [2] provides a good review of spectral methods.

3.4 Multi-level Approaches

An important improvement to FM has been to integrate *clustering* into a "two-phase" methodology. A k-way clustering of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is a set of disjoint clusters $P^k = \{C_1, C_2, \ldots, C_k\}$ such that $C_1 \cup C_2 \cup \ldots \cup C_k = \mathcal{V}$. where k is sufficiently large(usually $k = \theta(n)$). For ease of notation we will write the input hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ as $\mathcal{H}_0(\mathcal{V}_0, \mathcal{N}_0)$. A clustering $P^k = \{C_1, C_2, \ldots, C_k\}$ of $\mathcal{H}_0$ induces the *coarser* hypergraph $\mathcal{H}_1(\mathcal{V}_1, \mathcal{N}_1)$ with $\mathcal{V}_1 = \{C_1, C_2, \ldots, C_k\}$. For every $n \in \mathcal{N}_0$, the net n' is a member of $\mathcal{N}_1$ where $n' = \{C_i | \exists v \in n \text{ and } v \in C_i\}$, unless $|n'| = 1$, i.e., each cluster in n' contains at least one vertex in n. In two-phase FM, a clustering first induces the hypergraph H_1 from H_0, and then FM is run on $\mathcal{H}_1(\mathcal{V}_1, \mathcal{N}_1)$ to yield a partitioning $P_1 = \{X_1, Y_1\}$. This solution then projects to a new partitioning $P_0 = \{X_0, Y_0\}$ of $\mathcal{H}_0$, where $v \in X_0(Y_0)$ if and only if for some $C_h \in \mathcal{V}_1$, $v \in C_h$ and $C_h \in X_1(Y_1)$. Next, FM is run again on $\mathcal{H}_0(\mathcal{V}_0, \mathcal{N}_0)$ using P_0 as its initial solution. This second run can be classified as a *refinement* step, which refers to the idea that an initially good solution is further improved via local moves and swaps.

Many clustering algorithms for two-phase FM have appeared in the literature. Bui et al. [8] find a random maximal matching in the hypergraph and compact the matched pairs of modules into $\frac{n}{2}$ clusters; the matching can then be repeated to generate clusterings of size $\frac{n}{4}$, $\frac{n}{8}$,etc. Hagen and Kahng proposed a random-walk method in which cycles in the walk form the clusters [2]; Cong and Smith [15] compress cliques of modules into clusters; and Alpert and Kahng [2] cluster via graph traversals. All of these methods when used within two-phase FM significantly improve performance over standard FM. Further, two-phase FM is frequently faster than a single FM because the second phase starts with a good initial partition, and hence converges to a local minimum quickly.

The two-phase methods have been recently generalized to a *multilevel* approach leading to very successful graph partitioning tools Metis [35] and Chaco [31]. The general view of the multilevel approach is depicted in Figure 3.5. Recently, Borriello worked on applying the multilevel approach to hypergraphs [30]. Currently, PaToH (**Pa**rtitioning **To**ol for **H**ypergraphs), a multi-level hypergraph partitioning tool, is being implemented at Bilkent University.

In a multilevel algorithm, a clustering of the initial hypergraph $\mathcal{H}_0$ induces the coarsened hypergraph $\mathcal{H}_1$, then a clustering of $\mathcal{H}_1$ induces $\mathcal{H}_2$, etc. until the most coarsened hypergraph $\mathcal{H}_m$ is constructed. A partitioning solution $P_m = \{X_m, Y_m\}$ is found for $\mathcal{H}_m$ and this solution is projected to $P_{m-1} = \{X_{m-1}. Y_{m-1}\}$. P_{m-1} is then refined, e.g.,by using it as a starting solution for FM. This phase is called *uncoarsening* phase, and is continued until a refined partitioning of $\mathcal{H}_0$ is derived.

PaToH coarsens the graph by matching a vertex to the vertex that it shares most nets. Each vertex is matched for at most once in each coarsening level. The coarsened graph is partitioned and in the uncoarsening phase, the projected solutions are improved by an FM pass. This FM pass works only on the vertices adjacent to a net on the cut. Currently, PaToH makes multi-way partitions recursively. That is to partition a hypergraph into 4 parts, the hypergraph is partitioned into 2, and then each part is partitioned in to 2 once more.

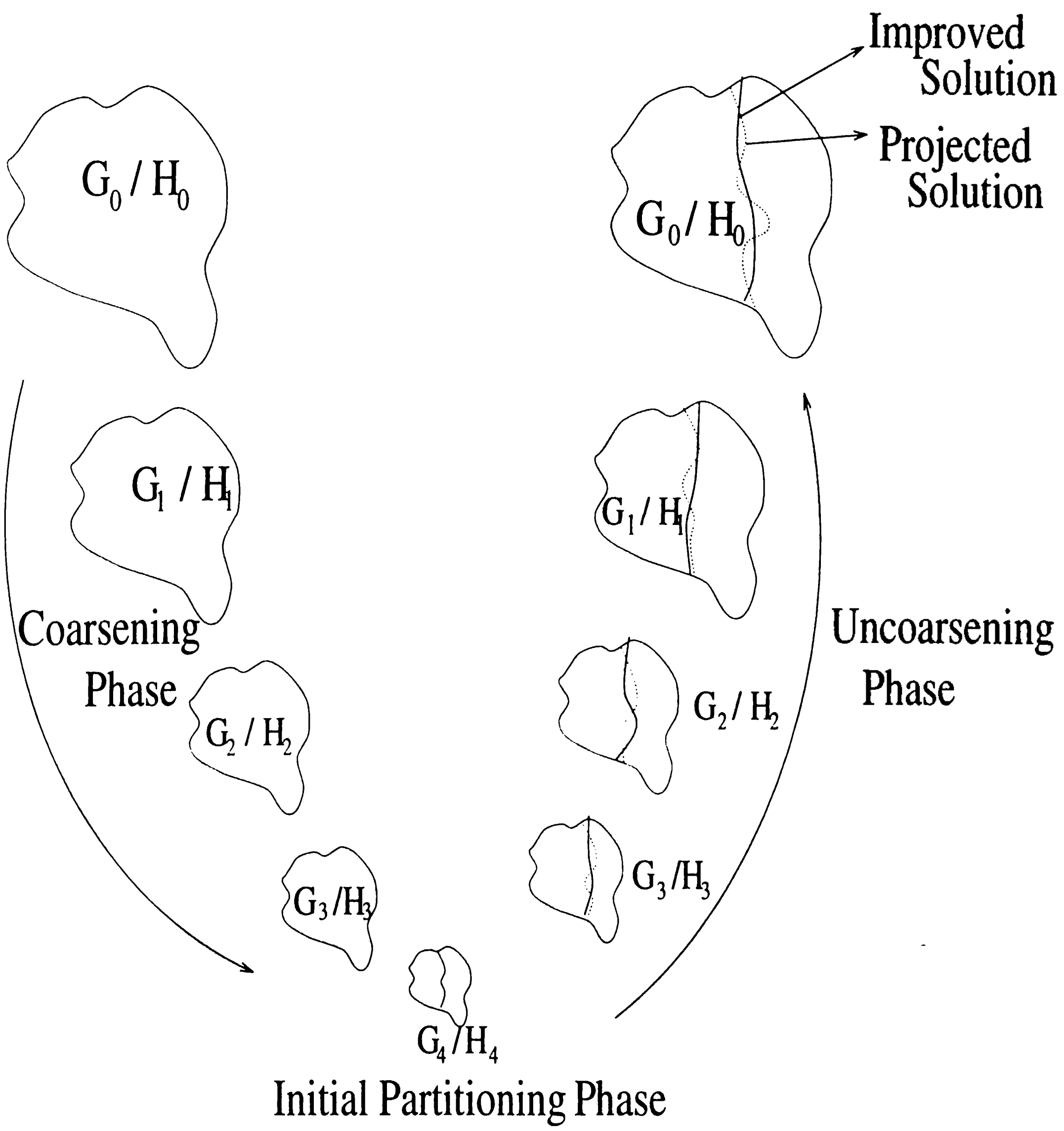


Figure 3.5. An overview of Multi-level Hypergraph Partitioning

4. Graph Partitioning by Node Separators

In this chapter, the problem of graph partitioning by node separators will be discussed. The problem is similar to graph partitioning by edge separators—discussed in the previous chapter—in the sense that it tries to identify logical clusters of nodes on the graph. However, this time it is a subset of vertices not the edges which separates the clusters of nodes. The problem has important applications in sparse matrix ordering, which aims at preserving sparsity during factorization.

The organization of this chapter is as follows: the first section defines the problem, the next section discusses some of its applications. Then we review previous work in the next section. The final section includes new ideas to find better separators.

4.1 Problem Definition

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph. Without loss of generality, we can assume the graph $\mathcal{G}$ to be connected. A vertex subset S is said to be a *node separator* if the subgraph induced by the vertices in $\mathcal{V} - S$ has more than one connected components. If the resulting graph has at least k connected components, then the set S is said to be a *k-way node separator* of $\mathcal{G}$. Formally, *k-way node separation* of $\mathcal{G}$ can be defined as follows:

Definition 4.1 *$\{P_1, P_2, \ldots, P_i, \ldots, P_k, S\}$ is a k-way node separation of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ iff the following conditions are satisfied:*

- $\forall i : 1 \leq i \leq k \quad P_i \not\equiv \emptyset$.
- $\forall i, j : 1 \leq i < j \leq k \quad P_i \cap P_j \equiv \emptyset$ *and* $\forall i : 1 \leq i \leq k \quad P_i \cap S \equiv \emptyset$

- $\bigcup_{i=1}^{k} P_i \cup S \equiv \mathcal{V}$
- $\nexists e = (u,v) \in \mathcal{E} \ni u \in P_i \ and \ v \in P_j \ and \ i = j$

A balance criterion can also be defined for the sizes of the parts. This balance criterion can be formalized as:

$$\frac{W_{max} - W_{avg}}{W_{max}} \leq \epsilon$$

where W_{max} is the weight of the part with maximum weight, W_{avg} is the average weight of parts (i.e., $W_{avg} = \frac{\sum_{v \in \mathcal{V}-S} weight(v)}{k}$), and ϵ is a predetermined imbalance ratio.

Using these definitions, the problem of partitioning by node separators can be stated as: "*Finding a balanced node partition with specified number of parts which minimizes the cardinality of the set S*".

4.2 Applications

An important application of finding node separators arises in sparse matrix ordering. Sparse matrix ordering aims at preserving sparsity in Gaussian elimination or Cholesky factorization. The basic step in this process is to partition the graph representing sparse symmetric matrix with node separators. The nodes are ordered as: nodes in the first block, nodes in the second block and the nodes in the separator. The process is repeated recursively for ordering the nodes within the blocks. This process is called *Nested Dissection*, which was proposed by George [26].

The problem of partitioning by node separators has also been successfully applied to hypergraph partitioning problems arising in VLSI design automation. Kahng *et.al.* [13] represents the hypergraph by a graph where each vertex corresponds to a net in the hypergraph. Then they try to find a partitioning of the hypergraph by finding a node separator on this graph.

4.3 Previous Work for Finding Node Separators

Previous work on finding node separations is limited to only two-way separations. The problem remains untouched for finding multi-way node separators. So, in this section we will review previous work for two-way separators. Two basic approaches have been proposed for finding a node separator in a graph. The first one consists of starting with an initial node separator, and then iteratively improving this separator. The second one is finding a good wide separator in the graph, and passing to a node separator using this wide separator. Pothen *et.al.* [50] present a good comparison of different methods proposed for the graph partitioning by node separators problem. We will review these two approaches in the following two sections.

4.3.1 Improving an Initial Separator

Liu proposed a method with two phases [42]. The first phase determines an initial separator based on the minimum degree algorithm, a popular reordering method in sparse matrix computation to reduce fillins during elimination. The next phase is an iterative process which improves the initial separator based on finding matchings on bipartite graphs.

Input: Graph $\mathcal{G} = (U \cup V \cup S, \mathcal{E})$ where $P = (U, V, S)$ is a node separation of $\mathcal{G}$.
Output: A new separator S' with $|S'| \leq |S|$

```
1.      Improved := true;
2.      while Improved do
2.1         if $|U| < |V|$ then      /*make $U$ the larger portion */
                interchange $U$ and $V$;
2.2         if a subset $Y$ of $S$ is found with $|Adj(Y,U) < |Y|$ then
2.2.1           $V := V \cup Y$;
2.2.2           $S := (S - Y) \cup Adj(Y,U)$;
2.2.3           $U := U - Adj(Y,U)$;
2.3         else
                Improved := false;
```

Figure 4.1. Algorithm for Improving an initial separator

The algorithm for the second phase of Liu's method is given in Figure 4.1. It depends on finding a subset Y of S, whose adjacency on one part, say U, contains fewer vertices than the set itself. Also Step 2.1 in Figure 4.1 prefers the larger part for matching to improve the balance between U and V. The critical point in this method is how to find the subset Y of the separator S, with $|Adj(Y, U)| < |Y|$. This set can be computed during searching for an augmenting path, which is the key point in finding a maximum matching on a bipartite graph. Below we will review the problem of finding maximum matchings on bipartite graphs. Then we will discuss how we can adopt this idea for improving an initial node separator.

4.3.1.1 Finding a Maximum Matching on a Bipartite Graph

We will start with the definition of a bipartite graph. A graph is bipartite if we can divide the set of vertices into two groups such that each edge is between vertices in different parts. Formal definition follows:

Definition 4.2 *A graph $G = (\mathcal{V}, \mathcal{E})$ is bipartite iff there exists $U \subset \mathcal{V}$ such that*

$$\forall\,(u, v) \in \mathcal{E}, u \in U \text{ and } v \in \mathcal{V} - U$$

Then we will call sets U and $\mathcal{V} - U$ a bipartition of $\mathcal{G}$.

Usually, we use $G = (U, \mathcal{V} - U, \mathcal{E})$ to denote that $\mathcal{G}$ is a bipartite graph with U and $\mathcal{V} - U$ being a bipartition.

A *matching* on a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a subset of its edges with no common endpoints. A vertex is *matched* if it is adjacent to an edge in the matching, and *unmatched* otherwise. The cardinality of a matching is the number of edges in it. A *maximum matching* is a matching of maximum cardinality.

A *path* is a sequence of vertices $\langle v_0, v_1, \ldots, v_{n-1}, v_n\rangle$ such that (v_i, v_{i+1}) is an edge for $i = 0, 1, \ldots, n-1$, and there are no repeated vertices. An *alternating path* is a path, with one edge in the matching M, and the next edge not in M. An *augmenting path* is an alternating path that begins and ends with unmatched vertices.

The asymptotically fastest algorithm known for finding a maximum matching on a bipartite graph is $O(\sqrt{\mathcal{V}}\mathcal{E})$ [33]. However, the algorithm described in Figure 4.2 usually has a better running time performance, although its asymptotic complexity is $O(\mathcal{V}\mathcal{E})$ [49].

```
Input: A Bipartite Graph $\mathcal{G} = (U, S, \mathcal{E})$
Output: A maximum Matching $M$ on $\mathcal{G}$.

Step 1 /* Initialize */
1.      $M = \emptyset$; /* Initialize matching */
2.      $X = \emptyset$;
Step 2 /* initial matching */
3.      for each vertex $v \in S$ do
3.1         if there exists an unmatched vertex $u \in U$, adjacent to $v$ then
3.1.1           $M = M \cup \{(u, v)\}$;
3.1.2       else
                $X = X \cup \{v\}$;
Step 3 /* Augment matching */
4       $X_{new} = \emptyset$ ; /* The set of unmatched vertices in S */
5       repeat /* perform one pass of the augmenting procedure */
5.1         Initialize all vertices as unvisited ;
5.2         for each vertex $v \in X$ do
5.2.1           search for an augmenting path from $v$,visiting only
                    vertices not visited in this pass;
5.2.2           mark all vertices reached as visited;
5.2.3           if an augmenting path is found then
                    Augment ( $M$, augmenting path)
5.2.4           else
                    $X_{new} = X_{new} \cup \{v\}$ ;
5.3         $X = X_{new}$;
5.4         $X_{new} = \emptyset$;
        until no augmenting path is found in a pass;
```

Figure 4.2. Algorithm for finding a maximum matching on a bipartite graph

The algorithm in Figure 4.2 depends on finding augmenting paths on a bipartite graph. The *initial matching* step finds a maximal matching on the graph (there are no edges that can be added to the current matching without violating the matching property). In the next step, we repeatedly search for augmenting paths, which help us to augment the size of the matching by one. The process of augmenting the matching via an augmenting path is depicted in Figure 4.4, and the algorithm is given in Figure 4.3.

Each pass of the augmenting procedure takes $O(\mathcal{E})$ time, and since each pass increases the size of the matching by 1, the number of passes is limited by

Input: A matching M and an augmenting path $Pt = \langle v_0, v_1, \ldots, v_{2i-1}, v_{2i}, \ldots, v_{2n}v_{2n+1}\rangle$
Output: Matching M augmented by path Pt

```
1.    for i=0 to n
1.1       $M = M \cup \{(v_{2i}, v_{2i+1})\}$
2.    for i=1 to n
2.2       $M = M - \{(v_{2i-1}, v_{2i})\}$
```

Figure 4.3. Algorithm for Augmenting a matching with an augmenting path

$min(|U|, |V|)$, which is an upper bound for the size of the matching. Hence the overall complexity of the algorithm is $O(min(|U|, |V|)\mathcal{E})$. Proof for the correctness of the algorithm can be found in [45]. Note that this asymptotic complexity, may not give a realistic view of the run-time behavior, because the number of augmenting paths is quite far away from $min(|U|, |V|)$, either because the size of the matching is fewer, and/or because it is possible to find a good initial matching with help of fast heuristics in the initial matching phase.

4.3.1.2 Using maximum matching to improve the separator

Liu's method is based on finding a subset Y of the separator S with $|Adj(Y, U)| < |Y|$. Liu adopts the idea of augmenting paths, and *alternating level structures*, which form the basis for finding maximum matchings on bipartite graphs. Liu's algorithm [42] does not necessarily find the maximum matching, but the algorithm terminates as soon as a set $Y \subseteq S$ with $|Adj(Y, U)| < |Y|$ is found. The search for set Y terminates, when a non-augmenting path is found from a vertex v. Alternating level structures rooted at v is formed and Y is chosen as the union of even levels in this alternating level structure. The set Y fulfills the requirements as we will discuss soon.

A formal definition for alternating level structures is given in [42] as follows:

Definition 4.3 *$L_0, L_1, L_2, \ldots, L_{2j-1}, L_{2j}, \ldots$ is an alternating level structure rooted at vertex v with respect to a matching M iff*

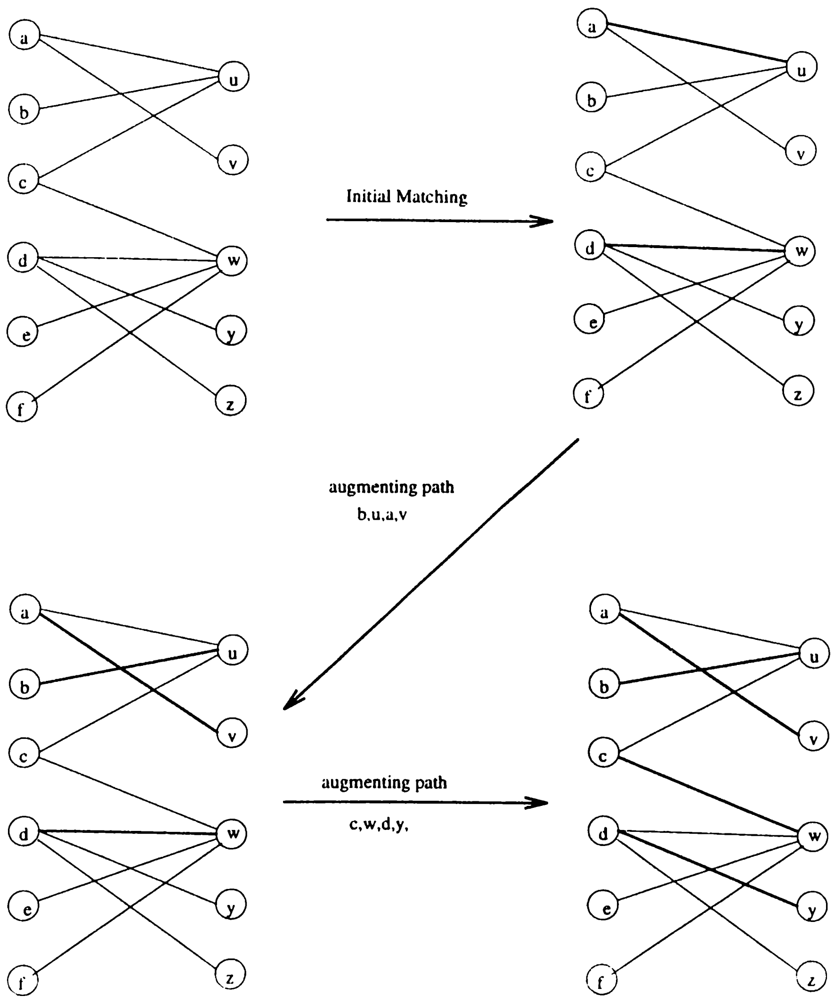


Figure 4.4. Improving matchings via augmenting paths

- $L_0 \equiv \{v\}$
- $L_{2j-1} \equiv Adj(L_0 \cup \ldots \cup L_{2j-2}) \quad for \;\; j = 1, \ldots$
- $L_{2j} \equiv \{u : \exists\, w \;\in L_{2j-1} \; and \, (u, w) \;\in\; M \quad for \;\; j = 1, \ldots$

If there is no augmenting path starting with vertex v, then there cannot be any unmatched vertex in the odd levels. This implies that each vertex in the odd level L_{2j-1} is matched to a vertex in L_{2j}. So we can say

$$|L_{2j}| = |L_{2j-1}|$$

Moreover, the level structure should end in an even level, because each vertex in the odd level L_{2j-1} is matched to a vertex. So L_{2j} is non-empty, if L_{2j-1} is non-empty. Let Y be the union of even levels,

$$Y \equiv L_0 \cup \mathrm{L}_2 \cup \ldots \cup \mathrm{L}_{2j} \cup \ldots$$

It can be verified that $Adj(Y, U)$ is the union of the odd levels. So we can conclude that

$$|Y| - |Adj(Y, U)| = 1$$

So the set Y, computed via alternating level structures fulfill the requirement for $Y \subseteq S$ and $|Y| < |Adj(Y, U)|$.

4.3.2 Finding a Node Separator from an Edge Separator

An edge separator can give us a good estimate of a node separator, since it finds logical clusters on the graph. We can exploit this fact and first find an edge separator and form a node separator by picking up vertices incident to an edge in the edge separator. Leiserson and Lewis defines the set of vertices incident to an edge on the cut as a ***wide separator*** [40]. This set forms a separator for the graph, because there are no edges between vertices in different parts after the removal of the vertices in the wide separator. The definition of a wide separator follows:

Definition 4.4 *Given a partition $P = (P_1, P_2, \ldots, P_i, \ldots, P_k)$ of graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, define $S_i \equiv \{v | v \in P_i$ and $(\exists(u, v) \in \mathcal{E} : v \in P_j$ and $i \neq j)$.*

The set

$$WS \equiv \bigcup_{i=1}^{k} S_i$$

is called a wide-separator *for graph $\mathcal{G}$ with respect to partition P.*

This wide separator forms disconnected components of the graph, but it is not optimal in the sense that it is always possible to form separators with smaller size, which we will call a *narrow separator.*

There are two key points in this approach: (i) How to find a good wide separator (ii) How to find a good narrow separator from the wide separator. The coming two sections discuss these points.

4.3.2.1 Finding Wide Separators

Finding a good wide separator is an important and difficult step for this approach. In fact the real difficulty is in the definition of the *goodness* for the wide-separator. We cannot state an objective which can give wide separators which always lead to better narrow separators. Minimizing the number of edges on the cut, may be a desirable criterion, since minimizing the number of edges may mean finding better logical clusters. But edge cuts with smaller cardinality can give worse wide separators.

Minimizing the number of vertices in the wide separator may be a good idea. Leiserson and Lewis [40] try to minimize the size of the wide separator by modeling the graph by a hypergraph. In their hypergraph model, each vertex in the graph is represented by a net in the hypergraph, and the respective net connects its representative and its adjacency. Formally, $\mathcal{H} = (\mathcal{C}, \mathcal{N})$ for representing the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as:

- $\mathcal{V} = \mathcal{C}$
- $n_i = \{v_i\} \cup Adj(v_i)$ for $i = 1, 2, \ldots, |\mathcal{V}|$.

The important characteristic of the hypergraph model is that vertices in the wide separator are exactly the representatives of the nets on the cut. Hence, minimizing the number of nets on the cut directly minimizes the size of the wide separator.

Minimizing the number of nodes in wide-separator is reasonable, but a wide separator with minimum number of nodes does not necessarily lead to a narrow separator of smaller cardinality, as we will discuss in Section 4.3.3.1.

4.3.2.2 From Wide Separators to Narrow Separators

As we have denoted above, it is always possible to find a subset of nodes in WS which forms a separator. Finding a narrow separator from a wide separator is equivalent to finding a vertex cover [16] on the graph induced by the wide separator. Since all edges are incident to a vertex in the vertex cover, the resulting set will be a good narrow separator for the whole graph, since there will be no edges among different parts (other than the separator).

Leiserson and Lewis [40] proposes three greedy heuristics for finding a narrow separator from a wide separator. The first one depends on repeatedly including the vertex with maximum number of edges on cut in the separator, and continuing until all edges in the cut are adjacent to a node in the node separator: The second heuristic depends on removing the vertex with minimum degree from the wide separator, and including all vertices adjacent to it in the narrow separator. Again this process continues until all edges on the cut are adjacent to a node in the node separator. Their third heuristic finds a trivial separator. If $P = (P_1, P_2)$ is a partition for a graph $\mathcal{G}$, they take the set $P_1 \cap WS$ or $P_2 \cap WS$ as the narrow separator, the cardinality of whichever is smaller.

However, Pothen ?? shows that we do not need to resort to heuristics, since the problem of obtaining a narrow separator form a wide separator can be solved optimally in polynomial time for bisections, by finding matching on the bipartite graph induced by vertices in the wide separator [50]. We start with vertices $S' \equiv P_1 \cap WS$ ($P_2 \cap WS$ if its cardinality is smaller) as the initial separator and recompute the separator by finding a maximum matching on the bipartite graph $BG = (S', P_2 \cap WS, \mathcal{E}_{cut}$. After finding the matching we decrease the size of the separator via alternating level structures rooted at unmatched vertices in the current separator, as discussed in Section 4.3.1.2. Note that unlike Liu's method we find the maximum matching only once. The iteration in Liu's method is not used.

4.3.2.3 Multi-way Node Separators

The problem of finding multi-way node separators remains untouched. Existing studies use recursive partitioning for multi-way separators. However, direct k-way partitions are usually faster, and give better results.

Trying to find a separator from a wide separator obtained by an edge-based partitioning seems reasonable. However, it is not possible to find a narrow separator optimally in polynomial time from a wide separator for multi-way separation. So, we need to resort to heuristics for this. The first two heuristics proposed by Leiserson and Lewis can be safely applied to this problem.

The first heuristic depends on moving the vertex with highest degree in the wide separator to the narrow separator. The process continues until all edges on the cut are adjacent to a vertex in the narrow separator. This heuristic, namely the *maximum removal* heuristic is presented in Figure 4.5.

Input: A Graph $\mathcal{G} = graph$.
Output: S = A vertex cover of $\mathcal{G}$.

```
1.    $S \leftarrow \emptyset$;
2.    repeat
2.1       $v \leftarrow$ vertex with maximum degree ;
2.2       $S \leftarrow S \cup \{v\}$;
2.3       $\mathcal{E} \leftarrow \mathcal{E} - Adj_{\mathcal{E}}(v)$;
      until ($\mathcal{E} = \emptyset$)
```

Figure 4.5. Algorithm for the Maximum Inclusion Heuristic proposed by Leiserson and Lewis

The second heuristic depends on removing the vertex with minimum degree from the wide separator, and including all vertices adjacent to it in the narrow separator. Again the process is repeated until all edges on the cut are adjacent to a node in the separator. This heuristic, namely the *minimum removal* heuristic is presented in Figure 4.6.

Both heuristics can be implemented with $\theta(\mathcal{E})$ asymptotic complexity by using a bucket-list data structure.

A similar problem arises in VLSI design automation , and is stated as *Module Allocation Problem* [14]. Our problem can be converted to this problem simply by replacing each edge on the cut by a module, and each vertex in the wide separator by a net. Cong et. al., proposed a solution for this problem by changing the problem to a maximum flow problem [14]. The fastest algorithm for this problem has $\theta(|V|^3)$ asymptotic time complexity, where $|\mathcal{V}|$ is the number vertices in the graph.

Input: A Graph $\mathcal{G} = graph$.
Output: S = A vertex Cover of $\mathcal{G}$.

```
1.      $S \leftarrow \emptyset$;
2.      repeat
2.1         $v \leftarrow$ vertex with minimum degree ;
2.2         for each $u \in Adj(v)$ do
2.2.1           $S \leftarrow S \cup \{u\}$;
2.2.2           $\mathcal{E} \leftarrow \mathcal{E} - Adj_{\mathcal{E}}(u)$;
        until $\mathcal{E} = \emptyset$
```

Figure 4.6. Algorithm for the Minimum Recover Heuristic proposed by Leiserson and Lewis

4.3.3 New Greedy Heuristics for Finding Separators

In this section, we will propose new methods for finding node separators.The first section, proposes a weightening scheme for finding better separators. The second section presents a new greedy heuristic for finding a narrow separator from a wide separator.

4.3.3.1 Finding Wide Separators

The difficulty in finding a good node separator has been discussed in Section 4.3.2.1. Neither minimizing the edges in the cut, nor minimizing the vertices in the wide separator are the right targets for us although they are valuable assets. We will propose a new method for finding a good separator in this section. This new method is based on the following two observations:

- A node with high degree is more likely to be on the separator.
- A node with a small degree is less likely to be on the separator.

As a corollary of this observation we can say that "*edges adjacent to a node with high degree are preferable to other edges to be in the cut*". Assume that half of the edges of a vertex are in the cut, then there is no merit in trying to recover

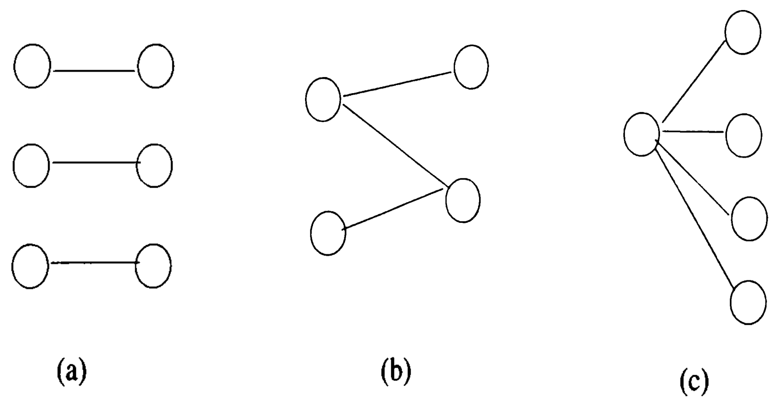


Figure 4.7. Three Different Wide separators

the rest of the edges of this node from the cut. On the other hand, recovering the only cut edge of a node is definitely a good step for a better separator.

Another point is that we want not only a small number of edges to be in cut, but also all edges in the cut to be adjacent to a small group of vertices. Three different wide separators are presented in Figure 4.7. The first one has minimum number of edges, and the second one has minimum number of vertices on the wide separator. The third one has neither minimum number of edges nor minimum number of edges, but it gives the best narrow separator, which is of size 1, since there is one vertex, which is adjacent to all edges on the cut. This example validates our argument that edges adjacent to a node with high degree are preferable to other edges to be on the cut.

For this purpose we propose a weightening scheme for the edges. In this scheme, each edge is assigned a weight disproportional to the degrees of its nodes. Let $e = (u, v)$ be an edge, the weight of this edge will be

$$weight(e) = \frac{1}{max(deg(u), deg(v))}$$

Minimizing the sum of weights of edges in the cut is expected to yield a better wide separator, since it favors cut edges adjacent to vertices with high degree.

4.3.3.2 From Wide Separators to Narrow Separators

In this section, we will propose a new greedy heuristic for finding a narrow separator from a wide separator. The heuristic is a hybrid of the maximum removal and save minimum heuristics, and is based on the observation stated in Lemma 4.1.

Lemma 4.1 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph and let $v \in \mathcal{V}$ and $deg(v) = 1$. There is always a minimum vertex cover $C \in \mathcal{V}$ of $\mathcal{G}$, where $v \notin C$.*

Proof: Assume the contrary. Let $v \in C$ with $deg(v) = 1$, where C is an optimal vertex cover of $\mathcal{G}$, and let u be the only neighbor of v. There are two cases:

(i) $u \in C$.
$C - \{v\}$ is still a vertex cover, and has a smaller cardinality, making a contradiction. Hence, $u \notin C$.

(ii) $u \notin C$
$C'' = C - \{v\} \cup \{u\}$ has the same cardinality as C, and is still a vertex cover.
So, there is always minimum vertex cover $C \in \mathcal{V}$ of $\mathcal{G}$, where $v \notin C$. ∎

Corollary 1 *Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph, and let $v \in \mathcal{V}$ be a vertex with degree 1, and let u be its only neighbor. Let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be the graph after the removal of u and v and edges adjacent to them and C' be a minimum vertex cover for $\mathcal{G}'$. Then $C' \cup \{u\}$ is a minimum vertex cover for $\mathcal{G}$.*

As a result of this corollary, we can say that the first step to find a vertex cover will be to include a vertex adjacent to a vertex of degree 1 in the vertex cover, and repeat this step as long as there exists a vertex with degree 1. By this way, we have the chance to obtain an optimal solution to the vertex cover problem, if we can find vertices with degree 1, until all edges are adjacent to a node in the node cover. However, we need to give a greedy decision when there are no vertices with degree 1. Actually, the minimum recover heuristic works in the same manner and it gives the greedy decision as saving the vertex with minimum degree, and moving all vertices adjacent to it to the separator. However, moving the node with highest degree to the separator seems to be a better greedy heuristic. Using this idea, we propose a new greedy heuristic namely *One-Max*, presented in Figure 4.8. This heuristic is a hybrid of the minimum recover and maximum inclusion heuristics, since it starts the same as the minimum recover heuristic and uses the same greedy decision as the maximum inclusion heuristic.

Input: A Graph $\mathcal{G} = graph$.
Output: S = A vertex Cover of $\mathcal{G}$.

```
1.      repeat
2.          for each $v \in \mathcal{V}$ $deg(v) = 1$ do
2.1             $u \leftarrow$ only neighbor of $v$
2.2             $S \leftarrow S \cup \{u\}$;
2.3             $\mathcal{E} \leftarrow \mathcal{E} - Adj_{\mathcal{E}}(u)$;
3.          if $(\mathcal{E} \neq \emptyset)$
3.1             $v \leftarrow$ vertex with maximum degree ;
3.2             for each $u \in Adjv(v)$ do
3.2.1               $S \leftarrow S \cup \{u\}$;
3.2.2               $\mathcal{E} \leftarrow \mathcal{E} - Adj_{\mathcal{E}}(u)$;
        until $\mathcal{E} = \emptyset$
```

Figure 4.8. Algorithm for a new greedy heuristic, One-Max

Other greedy heuristics may be subject to future research, but the first step in our heuristic, decreasing the problem size by making use of nodes with degree 1, should stay as it is.

5. Permuting a Sparse Matrix to Block Angular Form

The parallel solution of block angular Linear Programming (LP) problems has been a very active area of research in both operations research and computer science societies. One of the most popular approaches to solve block angular LP's is the Dantzig-Wolfe decomposition [18]. In this scheme, the block structure of the constraint matrix is exploited for parallel solution in the subproblem phase, where each processor solves a smaller LP corresponding to a distinct block. A sequential coordination phase (the master) follows. This cycle is repeated, until a suitable termination criteria are satisfied. Coarse grain parallelism inherent in these approaches has been exploited in recent research works. However, the success of these approaches depends only on the existing block angular structure of the given constraint matrix. The number of processors utilized for parallelization is clearly limited by the number of inherent blocks of the constraint matrix. Hence, these approaches suffer from *unscalability* and *load imbalance*.

This work focuses on the problem of decomposing irregularly sparse constraint matrices of large LP problems to obtain block angular form (BAF) with specified number of blocks for scalable parallelization. The objective in the decomposition is to minimize the size of the master problem while maintaining computational load balance among subproblem solutions. Minimizing the size of the master problem corresponds to minimizing the sequential component of the overall parallel scheme. Maintaining computational load balance corresponds to minimizing processors' idle time during each subproblem phase. So we can state our problem, computing the block angular form of a sparse matrix as: Finding a permutation of rows and columns of the matrix, to obtain a block angular form, with equal sized blocks, which minimizes the number of coupling rows.

This chapter proposes different methods for permuting a sparse matrix to block angular form. In each section, the graph model for representing the matrix will be presented first and will be followed by a discussion of algorithms to apply, and advantages and disadvantages of the model. The first subsection describes the work by Ferris and Horn [21], and the following subsections describe the models and solution methods we propose for computing the block angular form of a sparse matrix. We will make use of the following matrix in our examples throughout this chapter.

$$
\begin{array}{c}
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\
A = \left(\begin{array}{ccccccc}
x & & x & & & & \\
& x & & & & & x \\
x & x & x & x & & & \\
& x & & & x & & x \\
& & x & x & & & \\
& & x & x & & x & \\
& & & & x & x & x \\
& x & & & x & &
\end{array}\right)
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{array}
\end{array}
$$

Figure 5.1. The nonzero structure of the matrix A

5.1 Bipartite Graph Model

In this section, we will review the work by Ferris and Horn, which is based on a bipartite graph representation of the matrix. The first section explains the bipartite graph model, and the next section discusses the applicability of the model.

5.1.1 The Graph Model

In the bipartite graph Model (BG), a sparse matrix A is represented with a bipartite graph $BG = (R, C, \mathcal{E})$. Each row is represented by a vertex in R, and each column is represented by a vertex in C. For each nonzero a_{ij} in the matrix A, an edge is defined between the row vertex r_i and column vertex c_j. Clearly, the resulting graph is a bipartite graph with R and C forming the bipartition. A formal definition for the bipartite graph model is presented in Definition 5.1.

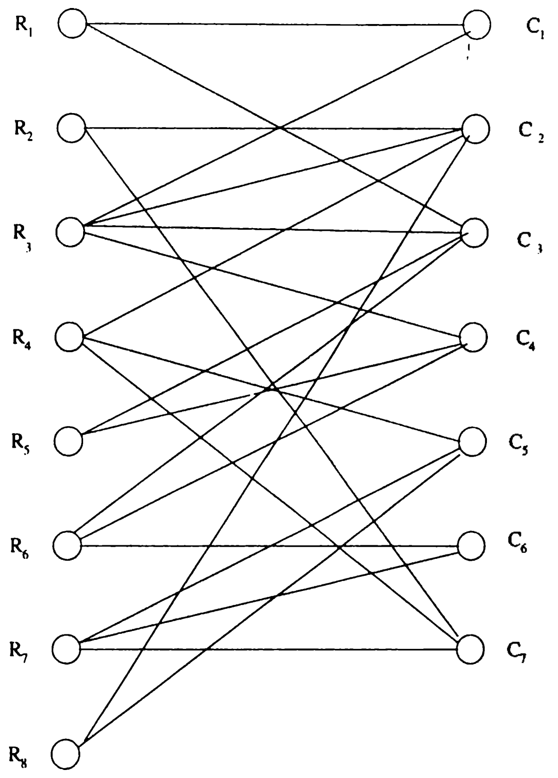


Figure 5.2. Bipartite Graph Representation of the matrix A in Figure 5.1

Definition 5.1 *A bipartite graph $BG = (R, C, \mathcal{E})$ is a BG representation of a sparse matrix $A_{M \times N} = (a_{ij})$ iff the following conditions are satisfied.*

- $\mathcal{V} \equiv R \cup C$ *and* $R \equiv \{r_1, r_2, \ldots, r_i, \ldots, r_M\}$ *and* $C \equiv \{c_1, c_2, \ldots, c_j, \ldots, c_N\}$, *where r_i and c_j represent the ith row and jth column of matrix A, respectively.*
- $e = (r_i, c_j) \in \mathcal{E}$ *iff* $r_i \in R$ *and* $c_j \in C$ *and* $a_{ij} \neq 0$

5.1.2 BAF with Bipartite Graph Model

Ferris and Horn partition this graph using the KL heuristic. They obtain a node separator from this graph using the maximal removal heuristic. This enables permutation of the graph into a doubly bordered block angular form. Out of the vertices in the separator, those representing the columns constitute the row-coupling constraints, and those representing the rows constitute the column coupling rows. This doubly bordered matrix can be transformed into a block angular matrix by *column splitting*, which will be discussed in Section 5.3.

Applying graph partitioning to the bipartite graph described above, naturally leads to computing the doubly-bordered block angular form of the matrix, because the vertices representing the columns and the vertices representing the rows are treated equally. Minimizing the size of the node separator in this graph corresponds to minimizing the sum of the number of rows and number of columns on borders in the doubly-bordered block angular matrix. However, our purpose is to find a block angular matrix, and the block angular matrix achieved by applying column-splitting on the dual block may be far away from the ideal block angular matrix, even if the doubly-bordered matrix were very close to an ideal case. So, decreasing the size of the node separator in the bipartite graph does not guarantee to decrease, and it may even increase the number of coupling rows in the resulting block angular matrix. This is basically because we are treating vertices of the bipartite graph equally for the operations on the graphs, but we have to treat columns and rows of the matrix separately.

So, we can not state a well-defined combinatorial optimization problem to establish a one to one relationship with the problem of permuting a sparse matrix into block angular form.

From the efficiency point of view, a major disadvantage of the bipartite graph model is that, it treats both columns and rows of the matrix as decision variables. This increase in the number of decision variables may be very costly, especially when $N >> M$. However it is possible to use either the rows or the columns of the matrix as decision variables and the other as the control variables. We will discuss such models in the following sections.

5.2 Row–Net Model

In this section, we will propose a new hypergraph model for representing a sparse matrix. In this model, each column is represented by a vertex, and each row is represented by a net in the hypergraph [48]. The first section describes our model, and the next section discusses how to use this model to find a block angular form

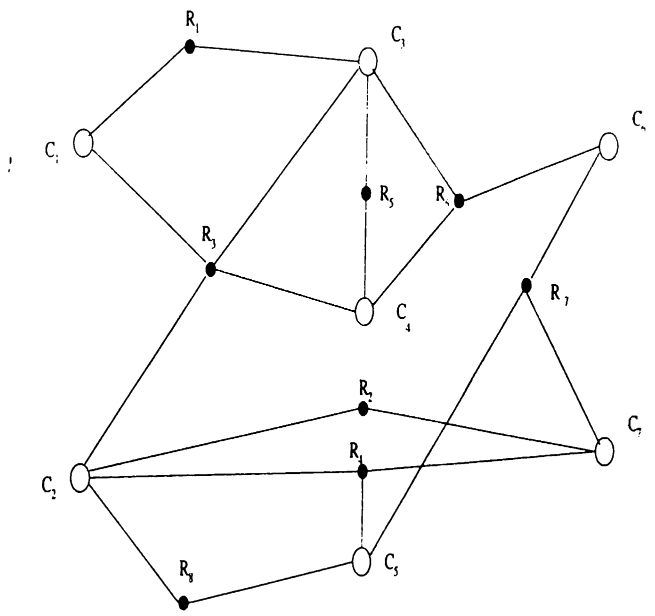


Figure 5.3. Hypergraph Representation of the matrix A in Figure 5.1 with Row-Net Model

of the sparse matrix.

5.2.1 The Hypergraph Model

In the row-net model, the matrix A is represented as a hypergraph $\mathcal{H}_{\mathcal{R}}(\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$. Each row is represented by a net (hyperedge) and each column is represented by a vertex in the hypergraph. There exist one vertex v_i and one net n_j for each column and row, respectively. Net n_j contains the vertices corresponding to the columns which have a nonzero entry on row j. Formal definition follows:

Definition 5.2 *A hypergraph $\mathcal{H}_{\mathcal{R}}(\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ is a row-net representation of a sparse matrix $A_{M \times N} = (a_{ji})$ iff the following conditions are satisfied.*

- $\mathcal{V} \equiv \{c_1, c_2, \ldots, c_i, \ldots, c_N\}$, *where c_i represents the ith column of matrix A.*
- $\mathcal{N} \equiv \{r_1, r_2, \ldots, r_i, \ldots, r_M\}$, *where r_i represents the ith row of matrix A.*
- $\forall v_i \in \mathcal{V}$ *and* $\forall n_j \in \mathcal{N} \ni \ v_i \in n_j$ *if and only if* $a_{ji} \neq 0$.

$$A_B^p = \begin{array}{ccccccc|c}
1 & 3 & 4 & 6 & 2 & 5 & 7 & \\
x & x & & & & & & 1 \\
 & x & x & & & & & 5 \\
 & x & x & x & & & & 6 \\
 & & & & x & & x & 2 \\
 & & & & x & x & x & 4 \\
 & & & & x & x & & 8 \\
x & x & x & & x & & & 3 \\
 & & & x & & x & x & 7
\end{array}$$

Figure 5.4. Block angular form of Matrix A in Figure 5.1

5.2.2 BAF with Row–Net Model

A k-way partition of $\mathcal{H}_R$ can be considered as inducing a row and column permutation on matrix A converting it into a block angular formA_B^p with k blocks as shown in Figure 5.3. Part P_i of $\mathcal{H}_R$ corresponds to block B_i of A_B^p. The set of external nets $\mathcal{N}_E$ corresponds to the coupling rows. That is, each cut net corresponds to a row of the submatrix $(R_1, R_2, \ldots, R_k)$ in Definition 2.4. Hence, minimizing the cutsize corresponds to minimizing the number of coupling constraints.

The hypergraph representation of the matrix A (Figure 5.1) with row-net model is illustrated in Figure 5.3. Let $P = (P_1, P_2)$ be a partition of the hypergraph with $P_1 = \{C_1, C_3, C_4, C_6\}$ and $P_2 = \{C_2, C_5, C_7\}$. This means: in the corresponding block angular matrix, columns 1, 3, 4, and 6 will be in the first block B_1, and columns 2, 5, and 7 will in the second block, B_2. This mapping of columns guides the mapping of rows. Nets R_1, R_5, and R_6 are internal nets in the first part, thus rows 1, 5, and 6 will be placed in the first part. Similarly, nets R_2, R_4, and R_8 are internal nets in the second part, thus rows 2, 4, and 8 will be placed in the second part. On the other hand, rows 3 and 7 constitute the coupling rows, since nets R_3 and R_7 are in the cut in the partition. After permuting rows and columns of the matrix A with respect to this partition we can obtain the block angular matrix illustrated in Figure 5.4.

Hypergraph partitioning finds a grouping of vertices in the graph, hence a grouping of the columns in the matrix. The grouping of rows is controlled by the grouping of columns. So, the decision variables in the hypergraph is limited by the columns in the matrix.

The objective of the hypergraph partitioning is to minimize the number of nets on the cut, while maintaining a certain balance criterion. Unlike, the bipartite graph model, there is a one to one correspondence between the problem of computing the block angular form of the matrix and the hypergraph partitioning problem. Maintaining balance between the part sizes in the graph corresponds to preserving the balance between the sizes of the blocks in the matrix. Minimizing the number of hyperedges on the cut corresponds to minimizing the number of coupling rows in the block angular matrix.

As discussed in Chapter 3, hypergraph partitioning problem has been studied heavily in VLSI design automation society, so it is possible to adopt heuristics from this domain. However, the performance of these heuristics directly depend on the characteristics of the input hypergraph. The hypergraphs representing sparse matrices might have significant deviations from hypergraphs representing circuits as in the case of LP constraint matrices. Matrices usually lead to dense hypergraphs compared to circuits, because there are physical limitations for the electrical components, which make the respective hypergraphs quite sparse.

KL-based methods are suitable for this application for their speed. However, these methods will suffer from the high net degrees, since it will be hard to save a net in the cut by a sequence of moves if the degree of the net is high. This makes KL-based methods heavily depend on the initial partition, since high degree nets are cumbersome for the iterative improvement process.

To handle such high degree nets, a multi-level approach, which shows an outstanding performance on graphs [35] might be useful. In the coarsening phase vertices are repeatedly matched, decreasing the size of the problem. It becomes easier to handle the partitioning in the small sized problem and an initial solution is obtained by this reduced hypergraph. In the uncoarsening phase, the initial solution is refined by unmatching the vertices step by step. This approach is more powerful in the sense that its dependency on the initial solution is less, and finds extremely better solutions compared to traditional approaches.

5.3 Column-Net Model

In this section, we will propose a second hypergraph model to find the block angular form. This model can be considered as the dual of the row-net model [48]. The first section describes the model, and the next section discusses how we can use this model, for finding a block angular form of a sparse matrix.

5.3.1 The Hypergraph Model

In the column-net model, the matrix A is represented as a hypergraph $\mathcal{H}_C(\mathcal{V}_\mathcal{R}, \mathcal{N}_C)$. Each column is represented by a net (hyperedge) and each row is represented by a vertex in the hypergraph. There exist one vertex v_i and one net n_j for each row and column of A, respectively. Net n_j contains the vertices corresponding to the rows which have a nonzero entry on column j.

Definition 5.3 *A hypergraph $\mathcal{H}_C(\mathcal{V}_\mathcal{R}, \mathcal{N}_C)$ is a column-net representation of a sparse matrix $A_{M\times N} = (a_{ij})$ iff the following conditions are satisfied.*

- $\mathcal{V} = \{r_1, r_2, \ldots, r_i, \ldots, r_M\}$*, where r_i represents the ith row of matrix A.*
- $\mathcal{N} = \{c_1, c_2, \ldots, c_i, \ldots, c_N\}$*, where c_i represents the ith column of matrix A.*
- $\forall v_i \in \mathcal{V}$ *and* $\forall n_j \in \mathcal{N} \ni \ v_i \in n_j$ *if and only if* $a_{ij} \neq 0$.

5.3.2 BAF with Column–Net Model

A k-way partition of $\mathcal{H}_C$ can be considered as converting A into a dual block angular form A_B^d with k blocks as shown in Figure 5.1. Part P_i of $\mathcal{H}_C$ corresponds to block B_i of A_B^d such that vertices and internal nets of part P_i constitute the rows and columns of block B_i, respectively. Each cut net corresponds to a column of the submatrix $(C_1^t, C_2^t, \ldots, C_k^t)^t$ in Definition 2.5.

The hypergraph representation of the matrix A with column-net model is illustrated in Figure 5.5. Let $P = \{P_1, P_2\}$ be a partition of this hypergraph with $P_1 = \{R_1, R_3, R_5, R_6\}$ and $P_2 = \{R_2, R_4, R_7, R_8\}$. Using this partition, rows 1, 3, 5 and 6 will form the first block B_1 , and rows 2, 4, 7, and 8 will form the second block B_2 in the corresponding block angular matrix. Nets C_1,

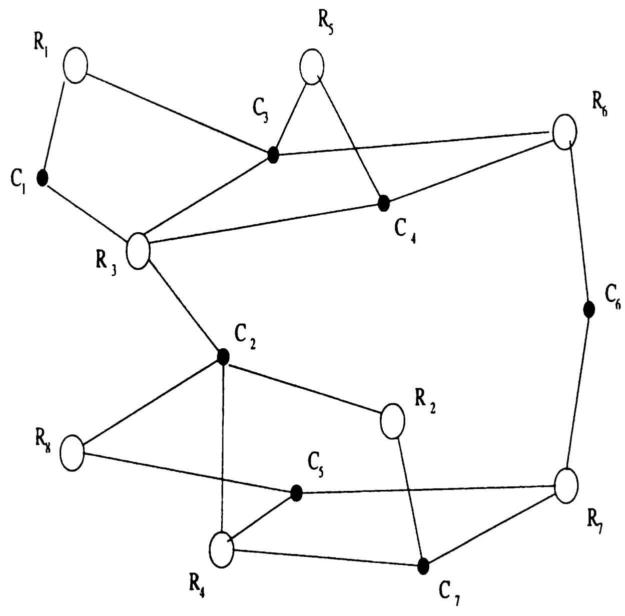


Figure 5.5. Hypergraph Representation of the matrix A in Figure 5.1 with Column-Net Model

$$A_B^d = \begin{array}{c} \begin{array}{ccccccc} 1 & 3 & 4 & 5 & 7 & 2 & 6 \end{array} \\ \left(\begin{array}{ccccccc} x & x & & & & & \\ x & x & x & & & x & \\ & x & x & & & & \\ & x & x & & & & x \\ & & & & x & x & \\ & & & x & x & x & \\ & & & x & x & & x \\ & & & x & & x & \end{array} \right) \end{array} \begin{array}{c} \\ 1 \\ 3 \\ 5 \\ 6 \\ 2 \\ 4 \\ 7 \\ 8 \end{array}$$

Figure 5.6. Dual block angular form of matrix A in Figure 5.1

C_3 and C_4 are internal nets in part P_1, thus columns 1, 3 and 4 will form the columns of the first block B_1. Similarly, columns 5 and 7 will be in the second block B_2, since nets C_5 and C_7 are internal nets in part P_2. Nets C_2 and C_6 are external nets, thus they will form the coupling collumns in the dual block angular matrix.

Dual block angular form of A_B^d leads to two distinct solution schemes. In the first scheme, we can apply special solution techniques for the dual block angular matrices. For example, if the matrix is the constraint matrix of a linear programming problem, we can exploit the fact that dual block angular constraint matrix of the original LP problem is a primal block angular constraint matrix of the dual LP problem. Hence, minimizing the cutsize corresponds to minimizing the number of constraints in the master problem of the dual LP. With this scheme, each net in the cut corresponds to a coupling row of the dual problem, hence minimizing the number of cut nets corresponds to minimizing the number coupling rows in the dual problem.

In the second scheme, we can transform the dual block angular matrix A_B^d into a primal block angular matrix by column–splitting as described in [21, 44]. This technique was originally proposed for multi-stage stochastic programs in linear programming, but it may be applied to all matrices representing a set of equalities or inequalities equally safely.

During the transformation process, For each column j of the submatrix $(C_1^t, C_2^t, \ldots, C_k^t)^t$, we introduce multiple column copies for the corresponding variable, one copy for each C_i that has at least one nonzero in column j. These multiple copies are used to decouple the corresponding C_i's on the respective variable such that the decoupled column copy of C_i is permuted to be a column of B_i. We then add column-linking row constraints that force these variables all to be equal. The process is depicted in Figure 5.3.2 and Figure 5.3.2. The column-linking constraints created during the overall process constitute the row-coupling rows of the matrix. With this scheme, Each column in the coupling block contributes "number of blocks it is connected -1" rows to the coupling block after transformation. The number of blocks a column is connected is equal to the connectivity of the respective net. Hence this time our objective in partitioning should be to minimize the connectivity, not the cutsize during the partitioning

$$A_B^d = \begin{pmatrix} x & x & & & & & & & \\ x & x & x & & & x & & & \\ & x & x & & & & & & \\ & x & x & & & & & x & \\ & & & & x & & x & & \\ & & & x & x & & x & & \\ & & & x & x & & & & x \\ & & & x & & & x & & \\ & & & & & 1 & -1 & & \\ & & & & & & & 1 & -1 \end{pmatrix}$$

Figure 5.7. Matrix A in Figure 5.6 after column-splitting

$$A_B^d = \begin{pmatrix} x & x & & & & & & & \\ x & x & x & x & & & & & \\ & x & x & & & & & & \\ & x & x & & x & & & & \\ & & & & & & x & x & \\ & & & & x & x & x & & \\ & & & & x & x & & & x \\ & & & & x & & x & & \\ & & & 1 & & & -1 & & \\ & & & & 1 & & & & -1 \end{pmatrix}$$

Figure 5.8. Block angular matrix A in Figure 5.6 after column splitting and permutation

process.

5.4 Row Interaction Graph

We will propose a third model in this section. In this model, each row is represented by a vertex in the graph, and edges in the graph represent the interaction between the respective rows [47]. The first section describes the new graph model, and the next section discusses solution methods with this new model.

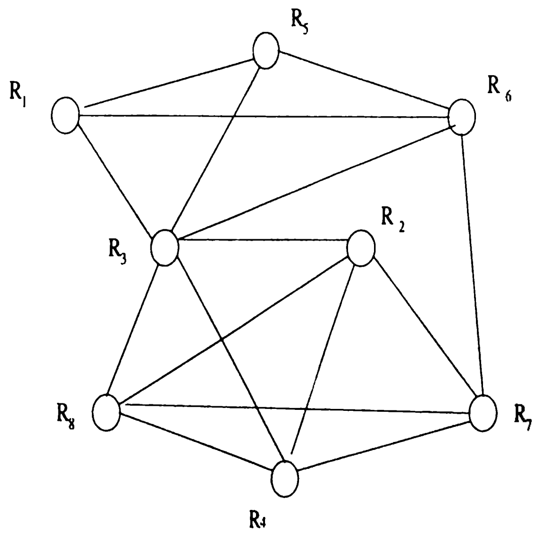


Figure 5.9. Row–Interaction Graph Representation of the matrix A in Figure 5.1.

5.4.1 The Graph Model

In the Row Interaction Graph (RIG), each row is represented by a vertex, and there exists an edge between two vertices if and only if the two rows have a nonzero in the same column. So formally:

Definition 5.4 *A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a RIG representation of a sparse matrix $A_{M\times N} = (a_{ij})$ iff the following conditions are satisfied.*

- $\mathcal{V} = \{r_1, r_2, \ldots, r_i, \ldots, r_M\}$*, where r_i represent the ith row of the matrix A.*
- $e = (r_i, r_j) \in \mathcal{E}$ *iff* $\exists k \; 1 \leq k \leq N \; \ni \; a_{ik} \neq 0$ *and* $a_{jk} \neq 0$

5.4.2 BAF with Row–Interaction Graph

By finding a node separator in the row interaction graph, we can induce a permutation of the rows and columns of the matrix into block angular form. After finding a separation $< P_1, P_2, \ldots, P_i, \ldots, P_k, S >$ for the row-interaction graph, vertices in the separator S correspond to the coupling rows, and vertices in part P_i correspond to the rows in block B_i. The permutation of the columns is controlled by the rows. Each column is placed in the same block as the rows it shares a non-zero with. By definition of the node separator, there are no edges between vertices in different parts, hence there is no interaction between rows in different blocks, i.e., there are no columns which has nonzeros in two rows at different parts.

The row interaction graph representation of the matrix A in Figure 5.1 is illustrated in Figure 5.9. We can find 2-way node separator $< P_1, P_2, S >$ of this graph as $P_1 = \{R_1, R_5, R_6\}$, $P_2 = \{R_8, R_2, R_4\}$ and $S = \{R_3, R_7\}$. Based on this separation rows $1, 5$ and 6 will form the first block, rows $8, 2$ and 4 will form the second block, and rows 3 and 7 constitute the coupling rows. Columns $1, 3, 4$ and 6 has nonzeros only in the first part, and in the separator, so we can place these columns in the first block. Similarly columns $2, 5$ and 7 are placed in the second block. The matrix permuted into block angular form with respect to this permutation is illustrated in Figure 5.4.

Just like row-net and column-net models, row interaction graph model enables us to state the problem of computing the block angular form of a sparse matrix, as a well-studied combinatorial optimization problem, finding node separators. The problem of graph partitioning by node separator. This problem has two objectives: (i) minimizing the number of vertices in the separator, (ii) maintaining balance between number of vertices in parts other than the separator. The first objective directly corresponds to minimizing the number of coupling rows, since each vertex in the separator correspond to a row in the coupling block. The second objective corresponds to load balance between the blocks.

5.5 Column–Interaction Graph

We can define a column interaction as a dual of the row interaction graph.

5.5.1 The Graph Model

In the Column Interaction Graph (CIG), each column is represented by a vertex. And there exists an edge between two vertices if and only if the two columns have a nonzero in the same row. So formally:

Definition 5.5 *A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a CIG representation of a sparse matrix $A_{M \times N} = (a_{ij})$ iff the following conditions are satisfied.*

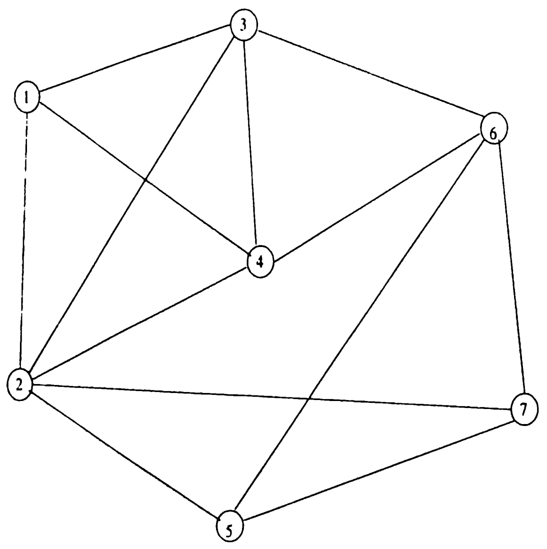


Figure 5.10. Column–Interaction Graph Representation of the matrix A

- $\mathcal{V} = \{c_1, c_2, \ldots, c_N\}$, *where* c_i *represents the* i*th column of the matrix* A.
- $e = (c_i, c_j) \in \mathcal{E}$ *iff* $\exists\, k \;\; 1 \le k \le N \;\; \ni \;\; a_{ki} \neq 0$ *and* $a_{kj} \neq 0$

5.5.2 BAF with Column–Interaction Graph

By finding a node separator in the column interaction graph, we can obtain a permutation for the dual block angular form of the matrix. In the resulting partition nodes in the separator form the coupling columns, and the other columns form the blocks. By definition of the node separator, there are no edges between vertices in different parts, and hence there is no interaction between columns in different blocks.

The resulting dual block angular matrix can be treated just in the same way as the dual block angular matrices obtained by the column–net model (discussed in Section 5.3).

A major drawback of this model is that the graphs representing the matrices are usually very dense. This directly depends on the density of the rows. So, it is both time-consuming and hard to find a good separator in CIG's of matrices, thus we have not included this model in our experiments.

6. Experimental Results

This chapter presents the results of various experiments we have performed to observe the validity of the proposed models. The first section describes the data sets we have used in our experiments. The next section describes our implementations. Then we present the results of our experiments with bipartite graph (BG) model, Row-Net (RN) model, Column-Net (CN) model, and the Row Interaction Graph (RIG) model, in turn, in separate sections. Each section presents the results of experiments of different methods we have used with these models. This chapter ends with a comparison of the performances and overall effectiveness of the models. The results presented in this chapter aims at giving an overview. Exhaustive presentation of these results can be found in the Appendix.

6.1 Data Sets

We have validated our models and associated methods on various Linear Programming Problems. Our first source for the data sets was *Netlib*. We have selected rather large problems out of the whole collection, since smaller problems do not need a parallel solution.

Our second source was *Kennington* problems again from Netlib. These problems are quite large, and form a good test bed for decomposition.

Our third source was the collection of Gondzio[1]. These problems are very large, and form a suitable test bed for us. The properties of all problems are presented in Table 6.1.

[1] These problems can be obtained by an anonymous ftp from IOWA Optimization Center
ftp col.biz.uiowa.edu
cd pub/testprob/lp/gondzio

Table 6.1. Properties of the Problems used in the Experiments
R, C, and Nz, represent the number rows, columns, and nonzeros respectively. $D\%$ represents the density of the matrix. t_{sol} shows the solution time (in seconds) of the problem by LOQO. C_{min} (R_{min}), C_{max} (R_{max}), and C_{avg} (R_{avg}) represent the minimum, maximum, and average number of nonzeros in a column (row). Numbers in the parentheses show the absolute minimum and the other represent the minimum greater than 1. Note that rows and columns of the matrix correspond to nets (vertices) and vertices (nets) in the hypergraph with RN (CN) model.

Problem	R	C	Nz	D%	C_{min}	C_{max}	C_{avg}	R_{min}	R_{max}	R_{avg}	t_{sol}
Netlib Problems											
25fv47	821	1571	10400	0.81	2(1)	21	6.62	2(0)	340	12.67	38.9
80bau3b	2262	9799	21002	0.09	2(0)	12	2.14	2(0)	112	9.28	90.8
bnl2	2324	3489	13999	0.17	2(1)	8	4.01	2(0)	82	6.02	188.6
cycle	1903	2857	20720	0.38	2(1)	28	7.25	2(0)	64	10.89	110.8
czprob	929	3523	10669	0.33	2(1)	4	3.03	2(0)	417	11.48	20.6
d2q06c	2171	5167	32417	0.29	2(1)	34	6.27	2(1)	144	14.93	400.0
ganges	1309	1681	6912	0.31	2(1)	13	4.11	2(1)	84	5.28	21.9
greenbea	2392	5405	30877	0.24	2(1)	24	5.71	2(0)	275	12.91	166.3
greenbeb	2392	5405	30877	0.24	2(1)	24	5.71	2(0)	275	12.91	115.3
scfxm3	990	1371	7777	0.57	2(1)	20	5.67	2(1)	57	7.86	13.8
sctap2	1090	1880	6714	0.33	2(1)	6	3.57	3(3)	24	6.16	9.8
sctap3	1480	2480	8874	0.24	2(1)	6	3.58	3(3)	31	6.00	12.2
ship12l	1151	5427	16170	0.26	3(1)	6	2.98	6(0)	75	14.05	20.5
ship12s	1151	2763	8178	0.26	3(1)	6	2.96	2(0)	49	7.11	10.4
sierra	1227	2036	7302	0.29	2(2)	4	3.59	2(2)	24	5.95	11.9
stocfor2	2157	2031	8343	0.19	2(1)	10	4.11	2(1)	15	3.87	24.8
woodw	1098	8405	37474	0.41	2(1)	21	4.46	2(1)	1477	34.13	80.7
Kennington Problems											
cre-a	3516	4067	14987	0.10	2(2)	14	3.69	2(0)	359	4.26	40.8
cre-c	3068	3678	13244	0.12	2(2)	13	3.60	2(0)	316	4.32	40.7
cre-d	8926	69980	242646	0.04	2(2)	13	3.47	2(0)	807	27.18	6719.9
osa-07	1118	23949	143694	0.54	6(6)	6	6.00	17(17)	17612	128.53	398.7
Gondzio Collection											
CO9	10789	14851	101578	0.06	2(1)	28	6.84	2(0)	440	9.41	1827.6
CQ9	9278	13778	88897	0.07	2(1)	24	6.45	2(0)	390	9.58	1664.39
GE	10099	11098	39554	0.04	2(1)	36	3.56	2(1)	47	3.92	907.6
NL	7039	9718	41428	0.06	2(1)	15	4.26	2(0)	149	5.89	699.2
mod2	34774	31728	165129	0.01	2(1)	16	5.20	2(0)	310	4.75	5383.34
world	34506	32734	164470	0.01	2(1)	16	5.02	2(0)	341	4.77	25819.68

The t_{sol} fields in Table 6.1 present the solution time with LOQO in seconds. The numbers in parentheses display the exact minimum, and the other number is the minimum greater than 1.

Some of these problems might already have a block angular structure, but they still form a good test bed for checking the validity of our heuristics. Note that even if these problems have a block angular structure, decomposition with respect to this structure is not possible, unless additional information about how to identify the the block structure (e.g., number of blocks, sizes of blocks) is provided. Also note that, we can find block angular structures with any number of blocks, not being restricted by the inherent number of blocks.

Note that columns and rows of the matrices correspond to the vertices (nets) and nets (vertices) of the hypergraph in RN (CN) model. Hence Table 6.1 also represent the characteristics of the hypergraphs we are working on. The characteristics of Row Interaction graphs for these matrices are presented in Table 6.2.

6.2 Implementation of the Algorithms

All algorithms described has been implemented in *C* programming language using the public *Gnu C* compiler. We have used an implementation of the Sanchis' algorithm, Metis (a graph partitioning tool implemented in University of Minnesota, which exploits the multilevel idea for graphs), and PaToH (a hypergraph partitioning tool, being implemented at Bilkent University which also exploits multilevel idea for hypergraphs).

We have used a *SUN Sparc 5* workstation in all our experiments.

6.3 Experiments with the Bipartite Graph Model

Obtaining a block angular matrix with bipartite graph model has two phase. In the first phase (the partitioning phase), the graph is partitioned, and in the second phase (separator phase), a permutation is obtained by including rows and columns to the set of linking rows and linking columns, respectively. We have tried different alternatives for the two phases. Below, we will discuss our results.

Table 6.2. Properties of the RIG's of Matrices used in the Experiments

R, C, and Nz fields denote the number of rows, columns, and nonzeros in the matrix, and E_{RIG} field denote the number of edges in the respective RIG. V_{min}, V_{max}, and V_{avg} represent the minimum, maximum, and average number of vertex degrees in the graph. Numbers in the parentheses show the absolute minimum, and the other number shows minimum degree greater than 1.

Problem	R	C	Nz	E_{RIG}	V_{min}	V_{max}	V_{avg}
25fv47	821	1571	10400	11074	1(0)	365	26.98
80bau3b	2262	9799	21002	10074	1(0)	61	8.91
bnl2	2324	3489	13999	13457	1(0)	64	11.58
cycle	1903	2857	20720	27714	1(0)	149	29.13
czprob	929	3523	10669	7072	2(0)	418	15.22
d2q06c	2171	5167	32417	26991	1(1)	118	24.87
ganges	1309	1681	6912	7656	1(1)	95	11.70
greenbea	2392	5405	30877	33841	1(0)	230	28.30
greenbeb	2392	5405	30877	33841	1(0)	230	28.30
scfxm3	990	1371	7777	8749	1(1)	75	17.67
sctap2	1090	1880	6714	5505	2(2)	39	10.10
sctap3	1480	2480	8874	7386	2(2)	50	9.98
ship12l	1151	5427	16170	10673	2(0)	77	18.55
ship12s	1151	2763	8178	5345	2(0)	50	9.29
sierra	1227	2036	7302	4936	3(3)	27	8.05
stocfor2	2157	2031	8343	12738	2(2)	37	11.81
woodw	1098	8405	37474	20421	1(1)	417	37.20
cre-a	3516	4067	14987	20748	3(0)	903	11.80
cre-c	3068	3678	13244	18905	3(0)	766	12.32
cre-d	8926	69980	242646	181670	3(0)	844	40.71
osa-07	1118	23949	143694	52466	37(37)	1082	93.86
osa-14	2337	52460	314760	113843	37(37)	2301	97.43
CO9	10789	14851	101578	119208	1(0)	706	22.10
CQ9	9278	13778	88897	106156	1(0)	701	22.88
GE	10099	11098	39554	51015	1(1)	114	10.10
NL	7039	9718	41428	49025	1(0)	360	13.93
mod2	34774	31728	165129	285068	1(0)	940	16.40
world	34506	32734	164470	273779	1(0)	971	15.87

6.3.1 Partitioning Phase

We have used two graph partitioning tools for the partitioning phase. First one is an implementation of the Sanchis algorithm. The second tool we have used was *Metis*, a multi level graph partitioning tool. We have used the maximum removal heuristic to obtain a node separator after partitioning.

We have not experimented with the Sanchis method with all our data sets, because it fails to find partitions, and because a subset of the data set is enough to see the huge difference between the two tools. We have taken 20 runs for each data. Figure 6.1 gives a comparison of the two tools both in terms of solution quality and run time efficiency.

As you can see from this figure, the two tools are not even comparable both in terms of solution quality and run time efficiency. The relative performance of Metis decreases as the number of parts increases, since Metis uses recursive partitioning for small graphs. A more detailed comparison can be found in Table A.2. Metis is superior to Sanchis for all matrices, and in the three fields. The difference in quality can be as high as 13 times (stocfor2), and running time can be 32 times faster (80bau3b). This is basically because Sanchis fails to identify natural clusters of the graph, as the graph gets denser, and the degrees of vertices vary in a large range. Metis overcomes this problem by coarsening the graph and identifying natural clusters in the first few levels of the uncoarsening phase.

The running times with Metis seem reasonable for most problems, however for problems with too many columns run time increases considerably, making the model impractical for many large problems. The quality of the solutions are feasible enough to favor a parallel solution for some of the problems, however for many problems solution quality is not good enough for an efficient parallel solution.

6.3.2 Separator Phase

We have experimented three greedy heuristics for the second phase. The three heuristics, namely maximum inclusion (MI), minimum removal (MR), and One-Max (OM) which have been described in Chapter 4. We have also used another method for finding a separator, which we will call *Trivial separator*. The set of all

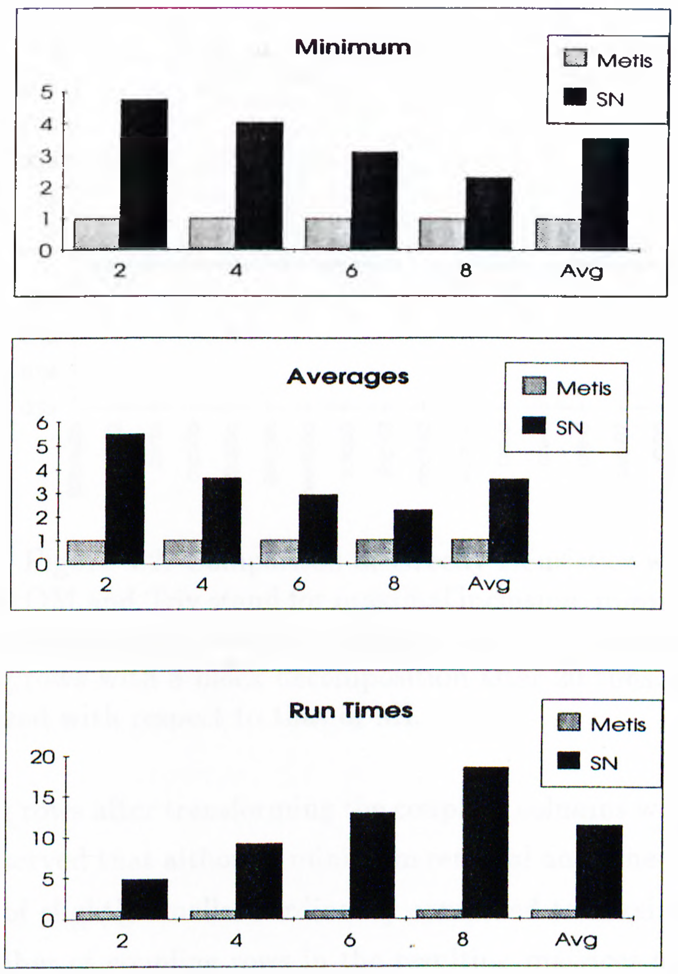


Figure 6.1. Comparison of Metis and Sanchis partitioning tools on BG model. Figure display results of 2,4,6, and 8 block decompositions. Minimum and average are results of experiments on 27 different matrices with 20 run for each, the numbers have been normalized with respect to that of PaToH.

row vertices, which are adjacent to an edge on the cut will give a separator, since vertices representing rows and vertices representing columns form a bipartition in the underlying graph.

We have experimented with the greedy heuristics for decomposition with 8 blocks and taken 20 runs for each data. The performance of the three greedy heuristics and the trivial separator method are presented in Figure 6.2.

The quality of the solutions, obtained by the three greedy heuristics are quite close to each other. The objective in this step is not finding a separator of minimum size, but finding a separator that is going to result in minimum number of

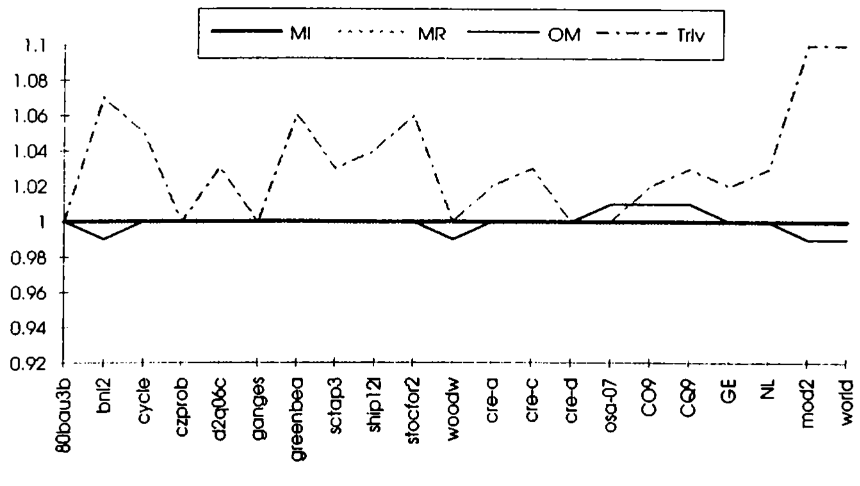


Figure 6.2. Comparison of Greedy Heuristics with BG Model
MI, MR, OM and Triv stand for maximal inclusion, minimum removal, One-Max, and trivial separator heuristics, respectively. The results are average number of coupling rows with 8 block decomposition after 20 runs. All numbers have been normalized with respect to that of MI.

coupling rows after transforming the coupling columns with column splitting. We have observed that although minimum removal and One-Max heuristics find separators of slightly smaller cardinality compared to maximum inclusion heuristic, the number of coupling rows in the resulting matrices may be more than those of maximum inclusion heuristic. Since linking columns can contribute more than one rows to the set of coupling rows, we can give priority to rows to be on the separator. Exploiting this, we have preferred to include rows to the separator as a tie-breaking scheme in all the greedy heuristics.

After all, the quality of the solutions, obtained by the three greedy heuristics are quite close to each other. This shows that the first part is the determinant for the performance of this model. Hence, real achievements on this model can be obtained by finding better partitions.

A remarkable point in Figure 6.2 is the performance of the trivial separator. Most of the time, this simple method was competitive with the greedy heuristics. It is 3% worse than greedy heuristics on the average, and the difference rises to 10% in the worst case. This shows that the rows of the matrix should have primary importance for decomposition. However, this model does not favor concentrating on rows, since a row and a column is represented equivalently by a vertex. Detailed results about this experiment can be found in Table A.3.

6.4 Experiments with the Row-Net Model

We have adopted two different hypergraph partitioning tools for our experiments with the row-net model, a Sanchis implementation and PaToH, a multi-level hypergraph partitioning tool. Our results show that partitioning tool is as important as the models we have proposed for the overall performance.

The quality of the solutions with Sanchis algorithm are not feasible for efficient parallelization. Most of the time, a significant part of the rows are placed in the coupling block making the parallel solution impractical. We can state the following observations for our experiments with the Sanchis' algorithm:

- For most matrices, there is a huge difference between the average case and best case performances. The difference between the average case and best case performances is quite typical for iterative improvement methods, because each run starts from a different initial partition. But the huge gap in our experiments means the performance of the Sanchis algorithm directly depends on the initial solution, and it fails to make good improvement on the initial solution for most of the cases.

- For many of the problems we have observed that the run-time of the heuristic decreases as the number parts increases, although not only the asymptotic complexity , but also run time for most practical applications increases with the number of blocks. The reason for this unexpected case is that Sanchis easily gets trapped in local optima, and performs very few passes, since the problem becomes harder as the number of parts increases.

- The quality of the solutions is expected to be better as we increase the number of levels in Sanchis' algorithm, since we increase the look-ahead ability of the algorithm. In our experiments increasing the number of levels slightly improved the solution quality, however the running time and especially the memory requirement increases enormously. The memory requirement increased to the order of G-Bytes for moderate sized problems, which make the algorithm impractical. Also, the achievements of increasing the look-ahead ability is not worth the increase in running time for most of the cases.

The results we have obtained with PaToH are definitely superior to those of Sanchis. A brief comparison of solution qualities and running times for Sanchis Level 1, Sanchis Level 2, and PaToH as a result of 20 runs is given in Figure 6.3. Details can be found in Tables A.4– A.6.

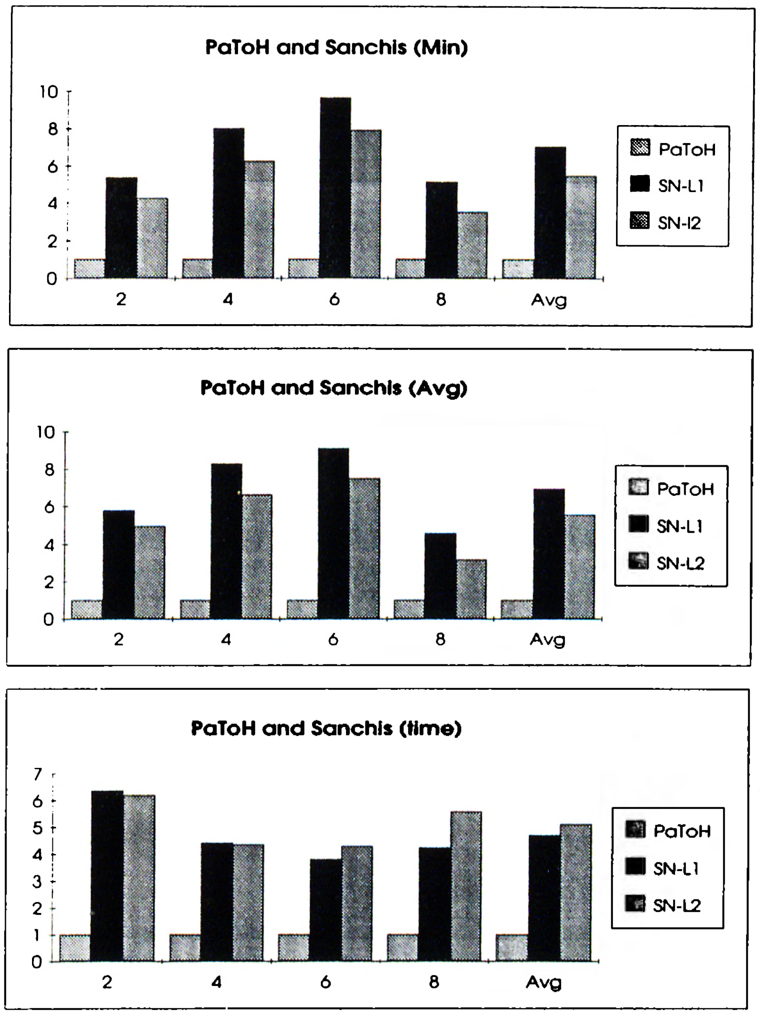


Figure 6.3. Comparison of PaToH and Sanchis (SN) for RN model
SN1 and SN2 represent the Level 1 and Level 2 of Sanchis' algorithm. The results show average difference on 27 different matrices with 2,4,6, and 8 block decompositions. Minimum and Average show the minimum and average number of coupling rows after 20 runs. Time is the average running time of 20 runs on 27 matrices. All numbers have been normalized with respect to that of PaToH.

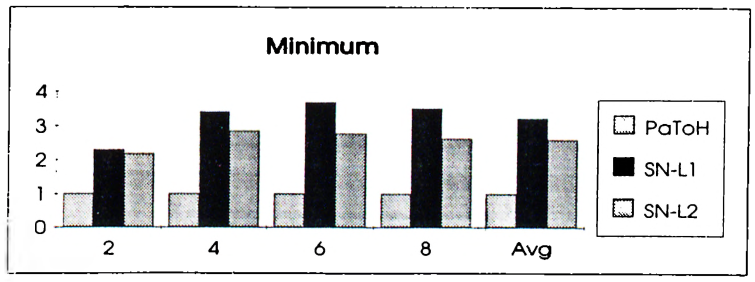
Moreover, these results are quite feasible for decomposing matrices for parallel solution both in terms solution quality and the running times. We were able to decompose a matrix with 10099 rows, 11098 columns, 39554 nonzeros into 8 blocks with only 596 coupling rows in 14.34 seconds and a matrix with 34774 rows, 31728 columns, 165129 nonzeros into 8 blocks with only 1310 coupling rows in 15.12 seconds. The solution times with *LOQO* are 907.6 seconds for the former and 5970.3 seconds for the latter.

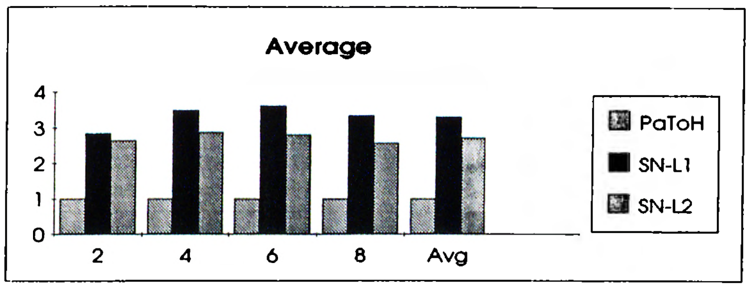
The main reason for the huge difference between the performance of Sanchis and PaToH is high net degrees. When a net with a high degree is on the cut, Sanchis' algorithm can not formulate the sequence of moves that will recover this net form the cut. The look-ahead ability in Sanchis' algorithm was proposed for such cases, however, that algorithm has been originally proposed for partitioning VLSI circuits, where the average net degrees are around 5. To handle high degree nets in the hypergraphs (the dense rows in the matrices) we need to use Level 10 or so in our experiments, but this is not possible since the memory requirement increases exponentially with the number of levels, and it reaches to the order of G-Bytes even with level 3 on moderate size problems.

PaToH repeatedly matches vertices, until the number of vertices drop down to order of hundreds, this decreases the size of the nets, and at this level we can identify natural clusters of the hypergraph (and the matrix), as we go up in the uncoarsening phase, the number of vertices and the sizes of nets increase, however what we need to do at these levels is just to slightly improve the solutions inherited from the previous level. The coarsening and uncoarsening process enables PaToH to escape from being blocked by dense nets.

6.5 Experiments with the Column-Net Model

The results of our experiments with the column-net model are quite parallel to those of row-net model, as we expected. The quality of the solutions of PaToH can be 10 times better than those of Sanchis (`Stocfor2` and 8 parts). The difference between run times is also very significant. In the few experiments where running time of Sanchis algorithm is competitive with PaToH, the differences between the qualities of solutions becomes drastic. Another drawback of Sanchis algorithm is its high memory requirement, we failed to partition the large matrices to 8 parts





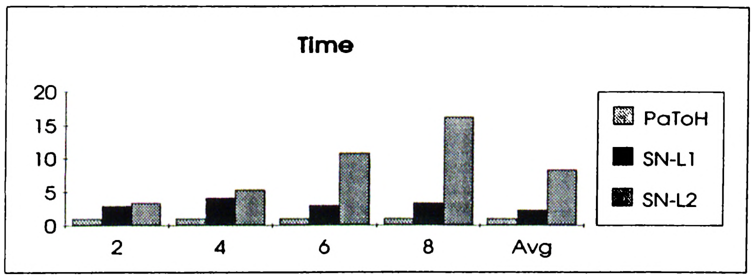


Figure 6.4. Comparison of PaToH and Sanchis (SN) for CN model
SN1 and SN2 represent the Level 1 and Level 2 of Sanchis' algorithm. The results show average difference on 23 different matrices with 2,4,6, and 8 block decompositions. Minimum and Average show the minimum and average number of coupling columns after 20 runs. Time is the average running time of 20 runs on 23 matrices. All numbers have been normalized with respect to that of PaToH.

with Level 2 of Sanchis' algorithm.

A brief comparison is given in Figure 6.4, while details are presented in Tables A.8 and A.9.

Column-net produced very poor results for some problems even with PaToH. (% 80 of columns has been left on the cut for `osa-07`, for 8 blocks). But it found fairly good decompositions for some problems, such as `ship08s`, `ship12s`, `stocfor2`, `80bau3b`, `cycle`, `ganges` and `scfxm3`. The performance of the Column-net model is relatively bad for large problems, such as `cre-d`, `osa-07`, `C09`, and `world`.

Parallel to the results of Column-Net model, Column-Net with transformation produces poor results for some problems. The results with PaToH after 20 runs are presented in Tables A.10–A.12. Number of coupling rows can be twice the original number of rows in the given matrix (`osa-07`, `cre-d`). However, the results are very promising for some of the problems, such as `stocfor2`, `ship08s`, `ship081` and `mod2`. Solution of the primal problem may be feasible for these problems.

The running time with PaToH is feasible enough to be considered as a preprocessing step for parallel solutions for large problems.

As a result, it might be useful to use the CN model, especially when other models do not work. Column-net with transformation might be useful if it is impractical to solve the dual problem, because of large number of columns.

6.6 Experiments with the Row Interaction Graph Model

We have only used Metis in our experiments, since our previous experiments show that multi-level approaches are definitely superior for our graphs.

The results with RIG model are quite promising. We were able to decompose a matrix with 10099 rows, 11098 columns, 39554 nonzeros into 8 blocks with only 517 coupling rows in 1.9 seconds and a matrix with 34774 rows, 31728 columns, 165129 nonzeros into 8 blocks with only 1029 coupling rows in 10.1 seconds. The solution times with *LOQO* are 907.6 seconds for the former and 5970.3 seconds for the latter.

Solutions with RIG model has two major steps: computing a wide-separator, and finding a good separator from the wide separator. We have applied different methods for the two parts of the problem. In the following two sections we discuss the experimental results of methods for the two parts of the problem.

6.6.1 Validity of Greedy Heuristics

We applied three greedy heuristics, maximum inclusion, minimum removal, and One-Max heuristics described in Section 4.3.2.3. We have also applied the maximum flow solution proposed by Cong et. al. [14]. We have modified the attraction function of a net stated in their study to $\frac{1}{C_M}$, where C_M denotes the number of modules on the cut this net is incident to, because the original function proposed in their paper produced extremely poor results.

Figure 6.5 gives a comparison of the performances of greedy heuristics and the maximum flow solution for this problem.

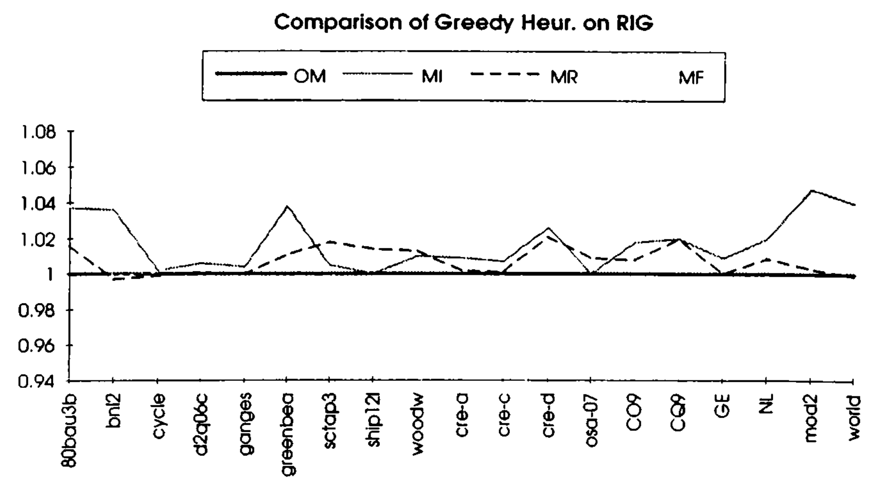


Figure 6.5. Comparisons of Greedy Heuristics for RIG
MI, MR, OM and MF denote the maximum inclusion, minimum removal , one-max and maximum flow solutions. Figure shows average number of coupling rows after 20 runs for 8-way partitions. All values have been normalized with respect to OM.

The table shows that the greedy heuristics find better separators than the maximum flow method. Among the three heuristics One-Max finds better results compared to the other two, but the difference is not that significant. Also the running time of the maximum flow solution is several times larger than the partitioning time, which makes it definitely useless for practical purposes.

In a second set of experiments, we have compared the performance of the heuristics with the optimal ones. Note that there is an optimal solution for finding a narrow separator from a wide separator for two way separators. This method is based on finding maximum matchings on bipartite graphs and is explained in Section 4.3.1.2. The results of our experiments are presented in Figure 6.6. The results show that greedy heuristics are very effective. The average improvement of optimal solutions is 0.12 percent, and it reaches a maximum of 0.58 percent in cre-d. These results show that we should concentrate on the first part of the method, finding good wide-separators for significant improvements with the RIG model. Detailed results for this set of experiments are presented in Tables A.13,

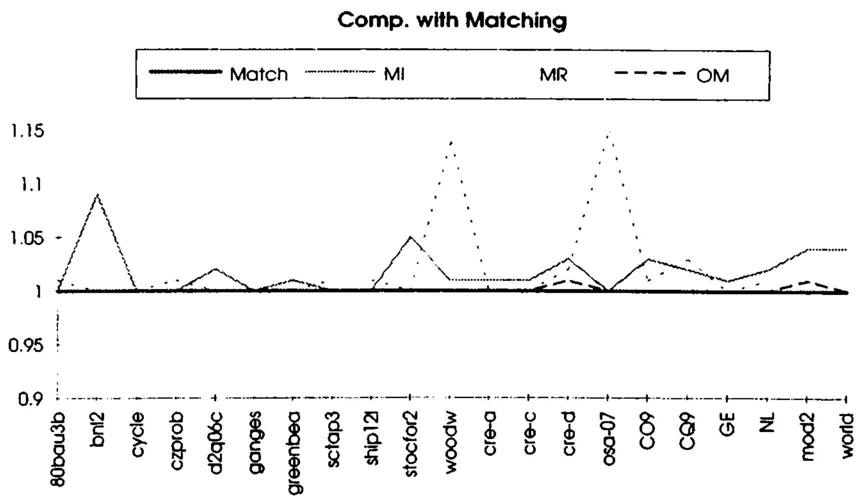


Figure 6.6. Comparison of Greedy heuristics with Optimal solutions

Match represent the optimal solution with matching, MI, MR, and OM represent the maximum inclusion, minimum recover, and One-Max heuristics, respectively. Figure shows average number of coupling rows after 20 runs for 8-way partitions. All values have been normalized with respect to that of Match.

and A.14.

6.6.2 Finding Wide Separators

We have used three different methods for finding wide separators. The first one is using direct graph partitioning and minimizing the number of edges on the cut. The second one minimizes the number of vertices in the wide separator by hypergraph partitioning as discussed in Section 4.3.2.1. The third method methods uses the weightening scheme discussed in Section 4.3.3.1. This method gives weights to edges and partitions the edge weighted graph. In our experiments we have discretized the weightening function, since Metis can only handle integer edge weights. An edge $e = (u, v)$ takes the weight value i if $max(deg(u), deg(v))$ falls into the ith segment, starting with the segment which contains the largest values. We have used six different weightening functions. They are :

- W1: - 30 -
- W2: - 25 - 50 -
- W3: - 20 - 40 - 60 -

- W4: - 15 - 30 - 45 - 60 -
- W5: - 10 - 20 - 30 - 40 - 50 - 60 -
- W6: - 20 - 40 - 60 - 80 -

Comparing different weightening schemes $W5$ produced the best results. So we take $W5$ as the representative of the weightening method. The results for these experiments can be found in Tables A.15- A.19.

Comparison of the three methods in terms of size of the minimum separator obtained in 20 runs, the average separator sizes and the run times are presented in Figures 6.7, 6.8 and 6.9, respectively. A more detailed comparison is given in Tables A.15- A.19. As expected, minimum number of edges in the cut

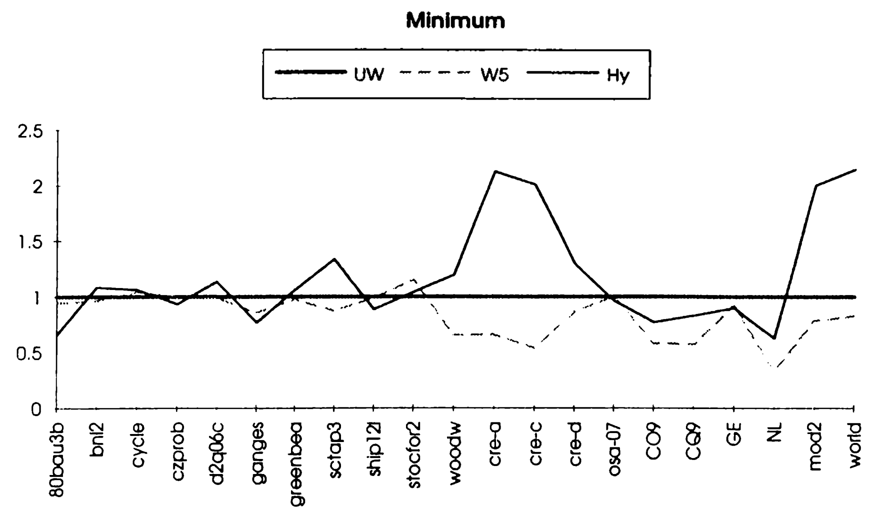


Figure 6.7. Comparison of Minimum Separator Sizes for different methods

UW represents finding wide separator by minimizing the edges on the cut. W5 is the weightening scheme explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. Figure shows a comparison of minimum separator sizes with 8 block partitions after 20 runs. All values have been normalized with respect to UW.

is obtained by direct graph partitioning and minimum number of vertices on the wide separator is obtained by the hypergraph model. However, weightening method gives better results for finding better separators, which is our primary objective. The differences in size of the resulting separators are quite significant for the three methods. We were able to find separators of 3 times smaller (NL and W5) with weightening compared to separator direct graph partitioning.

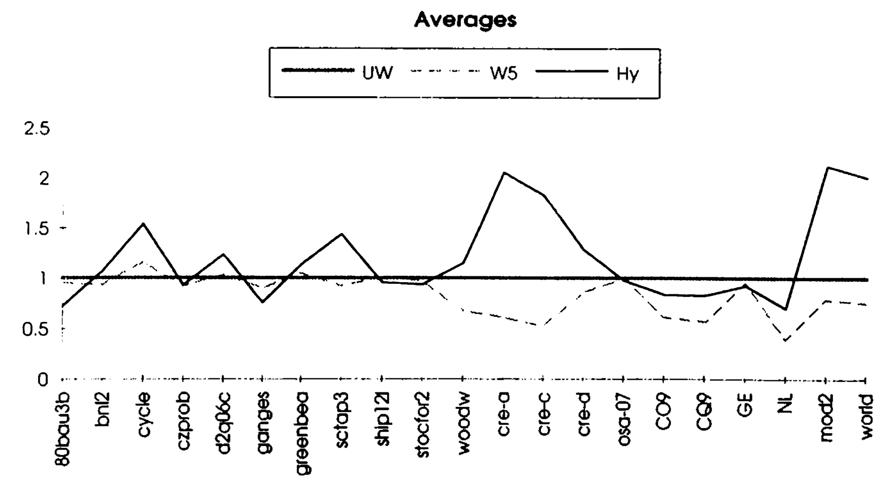


Figure 6.8. Comparison of Average Separator Sizes for different methods

UW represents finding wide separator by minimizing the edges on the cut. W5 is the weightening scheme explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. Figure shows a comparison of average separator sizes with 8 block partitions after 20 runs. All values have been normalized with respect to UW.

Comparing the difference between weighted and unweighted models. The quality of the solutions generated by the weighted model model are %14 better than the unweighted model. The run time of the weighted model is slightly better than the unweighted model. The results are presented in Tables A.22 and A.23

These results show that the quality of the wide separator has crucial importance in the performance of the RIG model, and there is much to be done for finding better separators.

6.7 Comparison of the Models

In this section, we will compare the performances of methods we have proposed. In our comparison, we have used the results of Metis and maximum inclusion heuristic for representing the bipartite graph model. Results of RN model are based on PaToH, and for RIG we have used weightening scheme 5 and One-Max heuristic. We did not include the CN model, in our comparison since it computes the dual block angular matrix, and we did not include CN with transfer, since it produces bad results for many of the problems. However, once again we want to point that CN with transfer finds very good results for some problems.

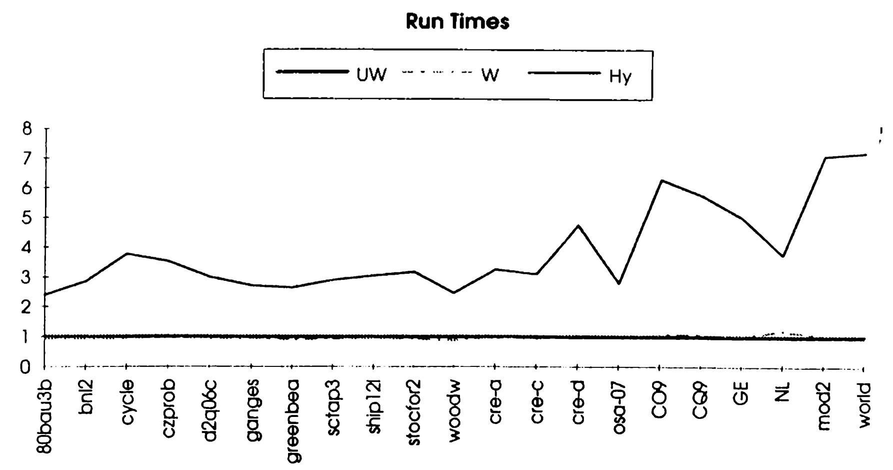


Figure 6.9. Comparison of Running Times for different methods

UW represents finding wide separator by minimizing the edges on the cut. W5 is the weightening scheme explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. Figure shows a comparison of running times with 8 block partitions after 20 runs. All values have been normalized with respect to UW.

We present the comparison of different models in Figures 6.10–6.13. Tables A.26–A.27 present a more detailed comparison.

Comparing RIG model with BG, we see that RIG overperforms BG for a great majority of the experiments in the quality of the solutions. In the four exceptions `scfxm3`, `ship12s`, `ship121` and `stocfor2`, the BG performs slightly better than RIG, especially for decomposing into small number of blocks. The relative success of BG model increases for decomposition with two blocks. This is simply because minimizing the size of the node separator in this graph is equivalent to minimizing the number of coupling blocks, since columns can not contribute more than one row to the set of coupling rows.

The difference between the performances of RIG and BG models becomes more significant for large sized problems. RIG is about twice as good as BG for the large problems (`cre-d`, `cre-c`, `cre-a`, `osa-07`, `NL`, `CQ9`), where decomposition is more necessary due to the increased problem size.

The quality of the solutions generated by RN model are competitive with those of RIG. In few exceptional cases differences become significant. RIG performs substantially better than RN on `czprob`, `osa-07` and `woodw`. On the other hand

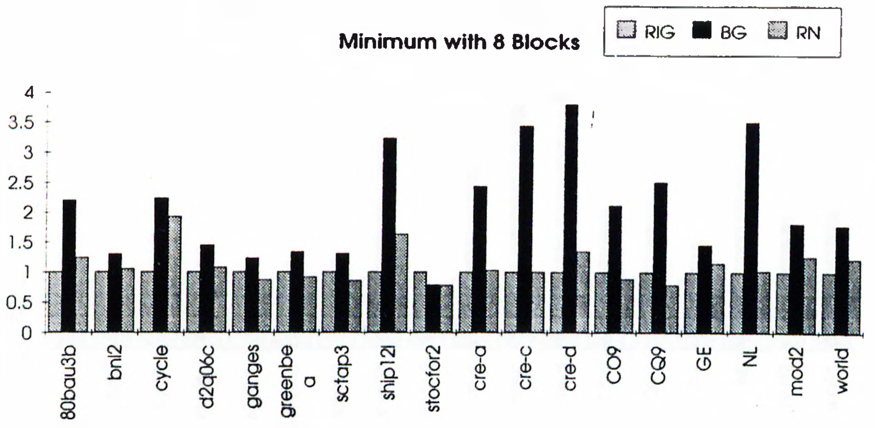


Figure 6.10. Comparison of best solutions of different models

Figure shows the minimum number of coupling rows with 8 block decomposition after 20 runs. All values have been normalized with respect to that of RIG.

RN defeats RIG on `ganges`, `sctap2`, `ship12s`, `ship12l` and `sctap3`. The differences are in the order of %10 for the rest of the results.

The results show that RN and RIG models defeat BG model. However, we can not say neither RN nor RIG outperforms the other, although the results of RIG is usually better than RN. The implementations used together with the models is as determinant as the models, as we have discussed all through the chapter, and it is hard to say the difference between RN and RIG origins from the model itself or the implementations used.

A remarkable point in the experiments is RIG performs significantly better than RN, when the number of columns increases. This is because RIG works with more compact information. However, RN has to deal with a lot of vertices. Most of these vertices give the same information for partitioning, but this information is very loose. This loose information is compacted in the RIG model, and the problem becomes much simpler. What we can conclude is that RN and RIG models successfully model the matrices for decomposition and reduce to a well-studied combinatorial optimization problem. These models enable an efficient parallel solution, based on decomposition.

Comparing the running times, RIG is the clear winner. It overperforms RN

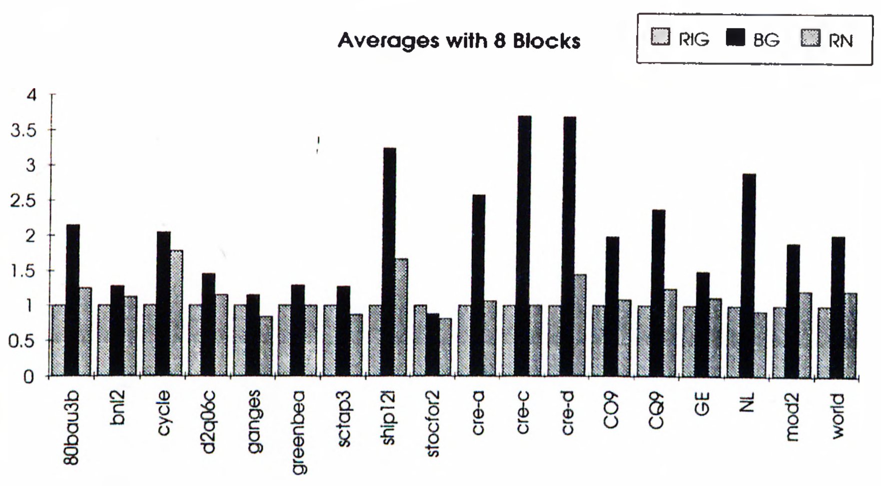


Figure 6.11. Comparison of averages of different models
Figure shows the average number of coupling rows with 8 block decomposition after 20 runs. All values have been normalized with respect to that of RIG.

and BG in all data sets. The only exception is on the BG model and NL problem. However, the quality of the solutions with BG model are quite poor compared to solutions with RIG model.

A remarkable point in running times is the enormous increase in the running times with BG model, with great number of columns. Running time of BG model is 22 times greater than running time of RIG model on cre-d. This makes use of BG model impractical for decomposing matrices with great number of columns, which is a common case for large matrices rising in Linear programming.

Finally, we want to show how effective the models are for decomposition. In Table 6.3, the number of coupling rows and the percent ratio of the number of coupling rows to the total number of rows, the actual partitioning times and percent ratio of partitioning times to solution times of the problems with *LOQO* [?] are presented. On the overall average, only 5.48% and 8.06% of the rows are on the coupling block for 4 and 8 block decompositions, respectively. The partitioning times are negligible compared to *LOQO* solution times (0.9% for 4 blocks, and 1.1% for 8 blocks). Another remarkable point in this table is that partitioning times grow slowly with the problem size, although solution times rapidly increase. This makes decomposition very practical for large problems.

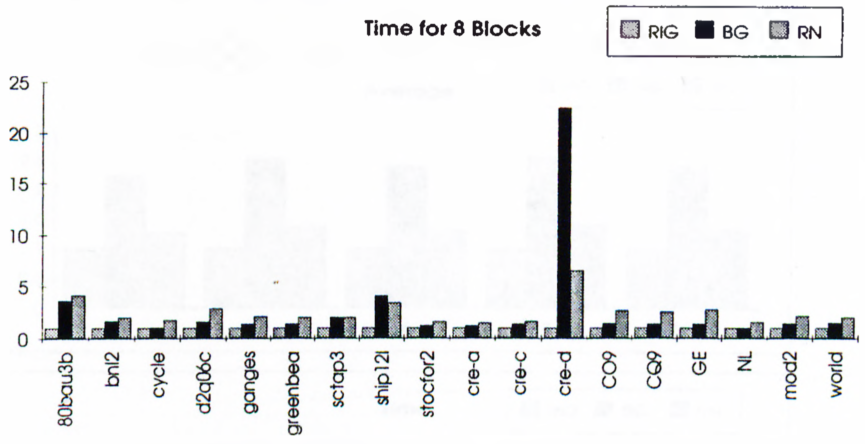


Figure 6.12. Comparison of running times for different models
Figure shows the minimum number of coupling rows with 8 block decomposition after 20 runs. All values have been normalized with respect to that of RIG.

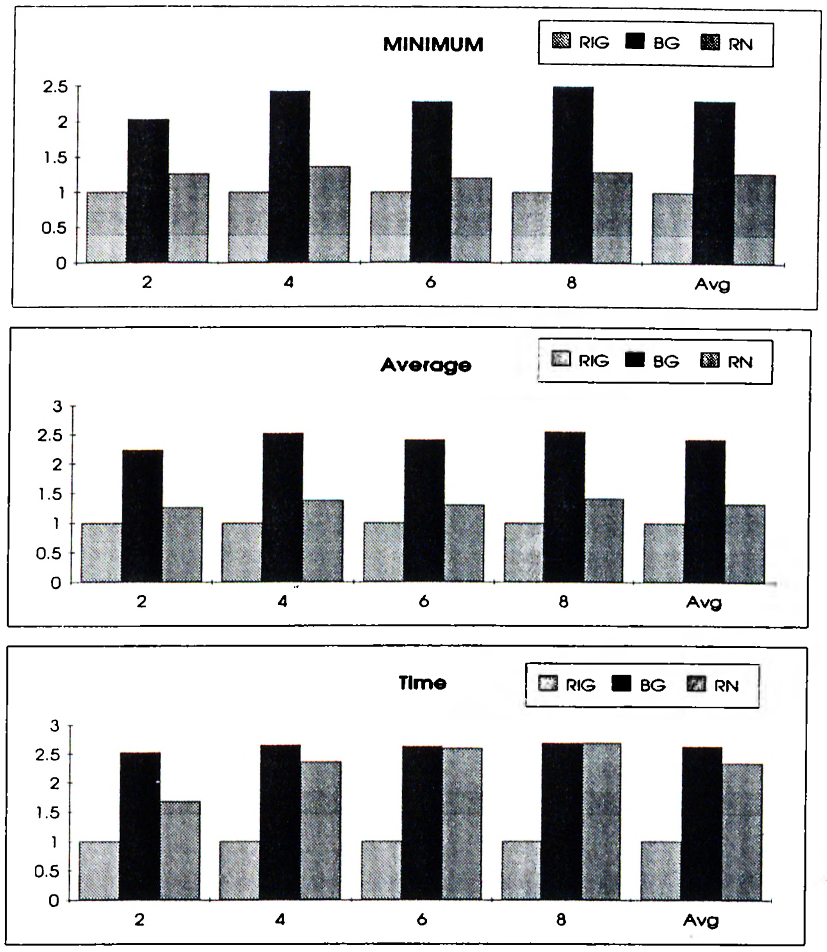


Figure 6.13. Figure gives a general comparison of different models for 2,4,6 and 8 block decomposition of 27 different matrices. The results after due to average of 20 runs. Values have been normalized with respect to RIG.

Table 6.3. The effectiveness of RIG Model

Problem			k	# Coup. Rows		t_{part}	
Name	Rows	LOQO $t_{sol}(secs)$		abs.	rel. %	abs. secs.	rel. %
cycle	1903	110.8	4	64	3.36	0.87	0.79
			8	100	5.25	1.05	0.95
d2q06c	2171	400.0	4	223	10.27	0.96	0.24
			8	293	13.50	1.17	0.29
ganges	1309	21.9	4	68	5.19	0.32	1.46
			8	128	9.78	0.41	1.87
greenbea	2392	166.3	4	125	5.23	1.34	0.81
			8	231	9.66	1.63	0.98
ship12l	1151	20.5	4	49	4.26	0.43	2.10
			8	78	6.78	0.54	2.63
stocfor2	2157	24.8	4	44	2.04	0.53	2.14
			8	120	5.56	0.66	2.66
woodw	1098	80.7	4	68	6.19	0.74	0.92
			8	160	14.57	0.86	1.07
cre-a	3516	40.8	4	112	3.19	1.03	2.52
			8	141	4.01	1.27	3.11
cre-c	3068	40.7	4	102	3.32	0.89	2.19
			8	127	4.14	1.08	2.65
cre-d	8926	6719.9	4	913	10.23	6.12	0.09
			8	1117	12.51	6.73	0.10
osa-07	1118	398.7	4	80	7.16	3.39	0.85
			8	80	7.16	4.05	1.02
CO9	10789	1827.6	4	1099	10.19	4.30	0.24
			8	1363	12.63	4.72	0.26
CQ9	9278	1664.4	4	751	8.09	4.00	0.24
			8	1061	11.44	4.36	0.26
GE	10099	907.6	4	331	3.28	1.71	0.19
			8	517	5.12	1.93	0.21
NL	7039	699.2	4	547	7.77	2.82	0.40
			8	633	8.99	3.22	0.46
mod2	34774	5383.3	4	559	1.61	9.44	0.18
			8	1029	2.96	10.07	0.19
world	34506	25819.7	4	615	1.78	9.24	0.04
			8	1074	3.11	10.02	0.04
Average			4		5.48		0.90
			8		8.06		1.1

7. Conclusion

In this chapter, we will discuss the results of our work, and give some directions for the future work.

7.1 Conclusions

This work focuses on the problem of permuting rows and columns of a sparse matrix into a block angular form, with specified number of equal sized blocks while minimizing the number of coupling rows. One major application of permutation into block angular form will be decomposing linear programs for parallel solution. However, note that the proposed solutions are valid for any sparse matrix, not only for constraint matrices of linear programs.

Many methods have been proposed for the parallel solution of block angular LP's. The general approach in these methods is exploiting the block structure of the constraint matrix for parallel solution in the subproblem phase where each processor solves a smaller LP corresponding to a distinct block. A sequential coordination phase (the master) follows. This cycle is repeated until suitable termination criteria are satisfied. However, the success of these approaches depends only on the existing *block angular* structure of the given constraint matrix. The number of processors utilized for parallelization in these studies is clearly limited by the number of inherent blocks of the constraint matrix. Hence, these approaches suffer from *unscalability* and *load imbalance*.

This work enables us to apply all methods proposed for problems with block angular problems, on any sparse problem, because we can permute the matrix into block angular form in a preprocess phase. Also this work enables us to work with any number of blocks, not being restricted by the inherent number of blocks.

In this study, we have proposed two hypergraph models, and a graph model to decompose matrices for parallel solution. We have experimented the validity of our models on various LP matrices, and compared our results with the bipartite graph model proposed by Ferris and Horn.

Our first model, namely the row-net model reduces the permutation problem to the well known hypergraph partitioning problem. We have experimented the validity of the model with various LP constraint matrices. We have used the algorithm of Sanchis and PaToH, a multi-level hypergraph partitioning tool being developed in our department, in our experiments. Our results show that the partitioning algorithm has a major importance in the validity of our model. We have observed that the quality of the solutions with Sanchis directly depend on the initial partition, and Sanchis' algorithm directly gets trapped in a local optima. We have obtained quite promising results with PaToH. The run times are quite fast and negligible compared to solution times. The quality of the solutions favor a solution by decomposition. The superiority of PaToH over Sanchis is basically because PaToH does not suffer from dense nets because of the coarsening process.

Our results with column-net are quite parallel to those of the row-net model. The difference between the performances of Sanchis and PaToH are quite similar to that of row-net model. Column-net usually leaves a great proportion of columns in the coupling block, however, the results for a subset of the problems are outstanding and favor parallel solution of the dual problem.

Column-net model with transformation usually produces large number of coupling rows, and sometimes twice the number of rows of the original problem. However, the results are very good for some of the problems, and it might be a good idea to solve the primal problem after transformation. The results show that, it is worth experimenting with the column-net model, especially when other methods do not produce good solutions.

The results of the RIG model are quite promising. The solution method has two phases. The first phase is finding a good wide separator, and the second phase is finding a narrow separator from the wide separator obtained in the first phase. Our experimental results show that the greedy heuristics we have proposed are good enough, and there can not be much more improvement for this phase, However, there is much to be done for finding a good node separator. There can be important deviations on the size of the separator by applying different methods on the first part of the problem. We have proposed a weightening scheme to find better wide separators. This method gives weights to edges of the graph, and partitions this edge weighted graph to find the separator. The results with this method is about 13 % better than classical methods, and give important key points for future work.

Comparing the quality performances of different models, RN and RIG models defeat the BG model. The difference becomes very significant for some problems. It is hard to compare RN and RIG models, since their performance is quite close to each other for all problems with a few exceptions. So, it is hard to say whether this difference originates from the models, or the implementations we have used in our experiments. The performance of RIG becomes superior to RN, as the number of columns in the matrix increases. Because RIG works on a more compact information, where as RN has to deal with lots of vertices.

RIG model is the clear winner in comparison of the running times. The running times of RN and RIG models are negligible compared to the solution times with LOQO. The running times with BG model can be high for problems with too many columns, which make the model impractical for such problems.

As a results, our models successfully model the matrices for decomposition and reduce the problem, to well-known combinatorial optimization problems. These models can be used to decompose matrices with very high quality (few coupling rows) on negligible time.

7.2 Future Work

This work gives many good spirits for future work. In this section we will briefly discuss some ideas which seem promising for future work.

7.2.1 Hypergraph Partitioning with Vertex Replication

As we have discussed above, hypergraph models we proposed reduce the problem to the hypergraph partitioning problem. Hypergraph partitioning tries to identify minimum subset of nets whose removal forms k disconnected equal sized parts.

That is it concentrates on only the nets to form disconnected components. Symmetrically, our RN model tries to identify a subset of rows whose removal forms independent blocks in the matrix. This model produces good decomposition if it is only rows to cause the interaction between the blocks. However, for some cases there are some columns (variables), making a lot of interaction between blocks causing a lot of rows to be on the coupling block. However, we can save many rows by splitting these columns. Column-splitting produces extra rows, increasing the overall problem size, however, the number of these extra rows may be less than the rows we will save by column-splitting. Bipartite graph model partially allows this, but it will be better to say it does not disallow, since there is no clever mechanism inside to handle this process.

A similar problem arises in FPGA partitioning in VLSI, and this problem has been stated as *hypergraph partitioning with module replication* in this society. The literature which addresses this problem is quite recent. It will be a good idea to exploit this idea for our problem, then we can say that our problem reduces to hypergraph partitioning with vertex (variable) replication problem, after modeling the matrices with the RN model.

7.2.2 Iterative Improvement Methods for Multi-way Separations

Our current work, for finding node separators includes only the two stage approach: finding a wide-separator, and moving to a narrow separator from this wide separator. an iterative improvement method, which takes an initial separator and then tries to decrease the size of this separator may increase the quality of our results. Liu proposed an iterative method for two way partitions. However, there is no multi-way generalization of this method. In Figure 7.1 we propose an iterative improvement algorithm for multi-way separation, which generalizes Liu's 2-way method.

In this algorithm, the subset Y can be computed by finding maximum matchings on the bipartite graph formed by S and $\mathcal{V} - \{S \cup P_j\}$, as explained in Section 4.3.1.2.

This algorithm not only decreases the size of the separator, but also helps us

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $P = (P_1, P_2, \ldots, P_k, S)$ is a node separation of $\mathcal{G}$.
Output: A new separator S' with $|S'| \leq |S|$

```
1.      Improved := true;
2.      while Improved do
2.1         Let $P_j$ be the part of minimum size
2.2         if a subset $Y$ of $P_j$ is found with $|Adj(Y, \mathcal{V} - \{S \cup P_j\}) < |Y|$ then
2.2.1           $P_j := P_j \cup Y$;
2.2.2           $S := (S - Y) \cup Adj(Y, V - \{S \cup P_j\})$;
2.2.3           $P_i := P_i - Adj(Y, P_i)$ for $1 \leq i \leq k$ and $i \neq j$;
2.3         else
                Improved := false;
```

Figure 7.1. Multi-way Separator Improvement Algorithm

to tune the balance among different parts.

7.2.3 Finding Coupling Rows after Partitioning on BG Model

In this section we will propose different methods for finding the set of coupling rows after the partitioning step in the bipartite graph model.

7.2.3.1 Finding a Separator of Minimum Size

Although Ferris and Horn finds a separator of minimum size after partitioning by a greedy heuristic, there is an optimal solution in polynomial time for this problem. This algorithm is based on finding maximum matchings on a bipartite graph. As we have discussed in Section 4.3.2.2 this problem has an optimal solution in polynomial time for bipartitions for general graphs. However, it is possible to find an optimal solution for multi-way partitions for this case by exploiting the fact that underlying graph is bipartite. We can start with an initial separator as all row vertices adjacent to an edge on the cut. Then we find a maximum matching on the bipartite graph formed by vertices adjacent to an edge on the cut. We can augment the separator using the method described in Section 4.3.1.2.

Please note that we will obtain minimum separators by this method, but these separators do not necessarily lead to minimum number of coupling rows.

7.2.3.2 Finding a Separator with Minimum Number of Coupling Rows

In this section, we will propose a new method to find a good separator in the bipartite graph model, which is going to result in minimum number of coupling rows. The problem can be stated as finding a vertex cover on the bipartite graph with minimizing the sum of weights of vertices, where weight of each row vertex is one, and weight of a column vertex is equal to its connectivity. The problem of finding a vertex cover on a bipartite graph with minimum sum of weights of vertices covered, can be solved by reducing the problem to *Maximum Flow* problem [16] by adding a source and destination vertex to the bipartite graph.

In this graph,

- The set of vertices formed by vertices adjacent to an edge in the partition and a source vertex s, and a terminal vertex t.
- There is an edge from the source s to all row vertices.
- There is an edge from all column vertices and the terminal t.
- The edges between row and column vertices are exactly those edges on the cut.
- All edges have unit cost.
- The capacity of all edges between the source vertex and row vertices, and edges between row and column vertices is equal to 1.
- The capacity of an edge between a column vertex and the terminal vertex t is equal to the connectivity of the column.

After computing the maximum flow on this graph, we can find the augmenting level structures on the bipartite graph (column and row vertices) as described in Section 4.3.1.2, starting from each unsaturated row vertex. After computing the level structures, the vertices in the even levels are replaced by vertices in the odd levels just like we do with maximum matchings.

This algorithm finds the optimal combination to produce minimum number of coupling rows, except for one point. It is not possible to compute the connectivity of a column safely. If all rows adjacent to a column in one block are moved to the separator, we will overcount the connectivity of this column. If we ignore this point, the algorithm described above optimally finds the solution for separators with minimum number of coupling rows.

Bibliography

[1] D. Adler. Switch-level simulation using dynamic graph algorithms. *IEEE Transactions on Computer-Aided Design*, 10(3):346–355, March 1991.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *Integration: the VLSI Journal*, 19(1–2):1–81, 1995.

[3] C. J. Alpert and S. Z. Yao. Spectral partitioning: The more eigenvectors, the better. In *Proc. ACM/IEEE Design Automation Conference*, 1995.

[4] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows.* John Wiley & Sons,Inc., 1990.

[5] J. F. Benders. Partitioning procedures for solving mixed variables programming problems. *Numerische Methamatik*, 4:238–252, 1962.

[6] K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial optimization. *Operations Research Letters*, 16:101–113, 1994.

[7] S. P. Bradley, A. C. Hax, and T. L. Magnanti. *Applied Mathematical Programming.* Addison Wesley, 1977.

[8] T. N. Bui, S. Chaudhuri, F. T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. *Combinatorica*, 7(2):171–191, 1987.

[9] T. Bultan and C. Aykanat. Circuit partitioning using parallel mean field annealing algorithms. In *Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 534–541, December 2–5 1991.

[10] T. Bultan and C. Aykanat. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, 16:292–305, 1992.

[11] U. V. Çatalyurek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplication. In *Proceedings of Irregular 96*, 1996. to appear.

[12] P. K. Chan, M. D. F. Schlag, and J. Y. Zien. Spectral K-way ratio-cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design*, 13(8):1088–1096, 1994.

[13] J. Cong, L. Hagen, and A. B. Kahng. Net partitions yield better module partitions. In *Proc. ACM/IEEE Design Automation Conference*, pages 47–52, 1992.

[14] J. Cong, W. Labio, and N. Shivakumar. Multi-way vlsi circuit partitioning based on dual net representation. In *Proceedings of IEEE Int. Conference Computer Aided Design*, pages 56–62, 1994.

[15] J. Cong and M'L. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 755–760, 1993.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1992.

[17] A. Daşdan and C. Aykanat. Improved multiple-way circuit partitioning algorithms. In *Proceedings of the ACM/SIGDA Second International Workshop on Field Programmable Gate Arrays*, pages 111–222, 1994.

[18] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[19] D. E. Van den Bout and T. K. Miller. Graph partitioning using annealed neural networks. *IEEE Transactions on Neural Networks*, 1(2):192–203, 1990.

[20] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications, 1990.

[21] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. Technical Report TR1232, Computer Sciences Department, University of Wisconsin Madison, May 1994.

[22] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[23] J. Frankle and R. M. Karp. Circuit placement and cost bounds by eigenvector decomposition. In *Proc. IEEE Intl. Conf. Coputer-Aided Design*, pages 414–417, 1986.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W.H. Freeman and Co., New York, New York, 1979.

[25] A. George and J. W. H. Liu. The evolution of the minimum degree ordering algortihm. *SIAM Review*, 31(1):1–19, March 1989.

[26] J. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[27] S. K. Gnanendran and J. K. Ho. Load balancing in the parallel optimization of block-angular linear programs. *Mathematical Programming*, 62:41–67, 1993.

[28] L. W. Hagen, D. J. H. Huang, and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on Computer-Aided Design*, 1995. Submitted.

[29] K. M. Hall. An r-dimensional quadratic placement algorithm. *Management Science*, 17:241–254, 1970.

[30] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. In *Proc. Chapel Hill Conf. on Adv. Research in VLSI*, 1995.

[31] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical Report SAND93-1301, Sandia National Laboratories, 1993.

[32] A. G. Hoffman. The dynamic locking heuristic – a new graph partitioning algorithm. In *Proceedings of IEEE International Symp. Circuits and Systems*, pages 173–176, 1994.

[33] J. E. Hopcroft and R. M. Karp. An $n^{(\frac{5}{2})}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[34] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[35] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[36] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[37] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[38] S. A. Kontogiorgis. *Alternating Directions Methods for the Parallel Solution of Large-Scale Block-Structured Optimization Problems.* PhD thesis, University of Wisconsin - Madison, 1994.

[39] B. Krishnamurthy. An improved min-cut algorithm for partitioning VLSI networks. *IEEE Transactions on Computers*, 33(5):438–446, May 1984.

[40] C. E. Leisserson and J. G. Lewis. Orderings for parallel sparse symmetric factorization. In *Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 27–31. SIAM, December 1987.

[41] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley and Sons, Inc.,, 1990.

[42] J. W. H. Liu. A graph partitioning algorithm by node separators. *ACM Transactions on Mathematical Software*, 15(3):198–219, September 1989.

[43] D. Medhi. Bundle-based decomposition for large-scale convex optimization: error estimate and application to block-angular linear programs. *Mathematical Programming*, 66:79–101, 1994.

[44] S. S. Nielsen and S. A. Zenios. A massively parallel algorithm for nonlinear stochastic network problems. *Operations Research*, 41(2):319–337, 1993.

[45] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity.* Prentice Hall, 1982.

[46] A. Pınar. A new genetic algorithm for hypergraph partitioning. In *Proceedings of TAINN'96*, pages 167–176, 1996.

[47] A. Pınar and C. Aykanat. A new graph model to decompose linear programs for parallel solution. *Lecture Notes in Computer Science.* To appear.

[48] A. Pınar, U. V. Catalyurek, C. Aykanat, and M. Pınar. Decomposing linear programs for parallel solution. *Lecture Notes in Computer Science*, 1041:473–482, 1996.

[49] A. Pothen and C. J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, December 1990.

[50] A. Pothen, H. D. Simon, and K. P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal on Matrix Analysis and Applications*, 11(3):430–452, July 1990.

[51] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer–Verlag, 1985.

[52] V. B. Rao and T. N. Trich. Network partitioning and ordering for mos vlsi circuits. *IEEE Transactions on Computers*, 6(1):128–144, January 1987.

[53] R. L. Rivest. The "pi" (placement and interconnect) system. In *Proceedings of the 19th Design Automation Conference*, pages 475–481. IEEE, 1982.

[54] L. A. Sanchis. Multiple-way network partitioning. *IEEE Transactions on Computers*, 38(1):62–81, Jan 1989.

[55] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proceedings of the 9th ACM/IEEE Design Automation Conference*, pages 57–62, 1972.

[56] A. Vanelli and K. R. Kumar. A method for finding minimal bottleneck cells for grouping part-machine families. *International Journal of Production Research*, 24:387–400, 1986.

[57] G. Vijayan. Partitioning logic on graph structures to minimize routing cost. *IEEE Transactions on Computer-Aided Design*, 9(12):1326–1334, December 1990.

A. Experimental Results in Detail

Table A.1. General Comparison of Sanchis (SN) and Metis
All values have been normalized with respect to Metis.

k	Min		Average		Time	
	Metis	SN	Metis	SN	Metis	SN
2	1.00	4.74	1.00	5.48	1.00	4.84
4	1.00	4.02	1.00	3.64	1.00	9.17
6	1.00	3.10	1.00	2.91	1.00	12.84
8	1.00	2.31	1.00	2.26	1.00	18.47
Avg	1.00	3.54	1.00	3.57	1.00	11.33

Table A.2. Comparison of Sanchis (SN) and Metis

Minimum and Average show the minimum and average number of coupling rows after 20 runs. Time is the average run time of 20 runs in seconds. Entries in the columns of Metis give the actual results and entries in the columns of SN have been normalized with respect to Metis.

Problem	k	Min		Average		Time	
		Metis	SN1	Metis	SN1	Metis	SN1
25fv47	2	92	1.17	123.15	1.75	0.42	3.90
	4	152	1.53	190.25	1.64	0.65	5.02
	6	213	1.39	244.60	1.41	0.79	6.44
	8	277	1.05	304.20	1.13	0.87	8.49
80bau3b	2	134	1.40	244.15	1.36	1.42	6.92
	4	437	1.30	536.85	1.36	2.25	15.23
	6	565	1.20	645.75	1.30	2.74	19.83
	8	699	1.07	733.90	1.16	2.96	32.89
bnl2	2	111	1.44	135.15	3.36	0.85	5.28
	4	277	1.59	310.60	2.04	1.11	11.77
	6	365	1.78	416.25	1.80	1.29	17.58
	8	509	1.35	542.70	1.47	1.39	21.32
cycle	2	58	4.98	69.90	6.38	0.75	4.51
	4	110	4.65	151.55	3.73	0.92	8.46
	6	141	3.77	156.15	3.82	1.01	10.78
	8	224	2.58	259.65	2.35	1.10	15.99
d2q06c	2	163	1.52	229.20	1.78	1.43	4.43
	4	311	1.62	352.35	1.68	1.63	10.34
	6	370	1.53	414.30	1.55	1.75	16.06
	8	424	1.37	466.05	1.42	1.85	22.67
ganges	2	27	2.59	36.00	5.99	0.30	4.70
	4	92	2.88	116.65	2.76	0.42	10.71
	6	134	2.22	159.80	2.21	0.51	14.73
	8	157	1.94	181.80	2.12	0.55	22.64
greenbea	2	94	9.14	130.55	8.10	1.58	4.73
	4	171	8.06	212.90	6.99	1.91	8.08
	6	212	7.04	302.35	5.28	2.08	11.53
	8	310	4.85	359.30	4.50	2.18	16.41
scfxm3	2	11	7.27	15.05	8.88	0.32	3.25
	4	28	6.61	35.85	6.13	0.47	5.23
	6	40	4.72	46.50	4.94	0.56	7.82
	8	87	2.29	102.00	2.32	0.64	10.48
stocfor2	2	14	13.14	20.40	11.74	0.47	5.87
	4	42	7.90	56.65	6.47	0.63	7.68
	6	95	4.26	112.45	3.87	0.72	10.83
	8	95	4.33	117.00	3.86	0.79	15.33

Table A.3. Comparison of Greedy Heuristics with BG Model
MI, MR, OM and Triv stand for maximal inclusion, minimum removal. One-Max, and trivial separator heuristics, respectively. Minimum and Average are the minimum and average number of coupling rows with 8 blocks after 20 runs. Columns of MI present the actual values and other columns have beeen normalized with respect to these columns.

Problem	Min				Average			
	MI	MR	OM	Triv	MI	MR	OM	Triv
25fv47	277	0.99	0.99	1.03	304.20	1.00	1.00	1.04
80bau3b	699	1.00	1.00	1.01	733.90	1.00	1.00	1.00
bnl2	509	0.98	0.98	1.07	542.70	0.99	0.99	1.07
cycle	224	1.00	1.00	1.02	259.65	1.00	1.00	1.05
czprob	455	1.00	1.00	1.00	455.50	1.00	1.00	1.00
d2q06c	424	1.00	1.00	1.05	466.05	1.00	1.00	1.03
ganges	157	1.00	1.00	1.00	181.80	1.00	1.00	1.00
greenbea	310	1.01	1.01	1.04	359.30	1.00	1.00	1.06
greenbeb	321	0.98	0.98	1.06	367.45	1.00	1.00	1.06
scfxm3	87	1.00	1.00	1.02	102.00	1.00	1.00	1.03
sctap2	204	1.01	1.01	1.00	225.05	1.00	1.00	1.02
sctap3	247	1.00	1.00	1.02	263.05	1.00	1.00	1.03
ship12l	252	1.00	1.00	1.02	254.00	1.00	1.00	1.04
ship12s	162	1.00	1.00	1.07	173.00	1.00	1.00	1.06
sierra	166	1.00	1.00	1.00	179.65	1.00	1.00	1.00
stocfor2	95	1.00	1.00	1.12	117.00	1.00	1.00	1.06
woodw	810	0.99	0.99	1.00	935.45	1.00	0.99	1.00
cre-a	344	1.00	1.00	1.01	399.40	1.00	1.00	1.02
cre-c	438	1.00	1.00	1.03	553.90	1.00	1.00	1.03
cre-d	4258	1.00	1.00	1.00	4597.55	1.00	1.00	1.00
osa-07	929	1.01	1.01	1.00	1025.60	1.01	1.01	1.00
CO9	2901	1.01	1.01	1.02	2987.40	1.01	1.01	1.02
CQ9	2678	1.00	1.00	1.03	2774.60	1.01	1.01	1.03
GE	757	1.00	1.00	1.01	864.30	1.00	1.00	1.02
NL	2221	1.00	1.00	1.03	2345.80	1.00	1.00	1.03
mod2	1874	0.99	0.99	1.09	2283.90	0.99	0.99	1.10
world	1921	1.00	1.00	1.10	2455.50	0.99	0.99	1.10
Average	1.00	1.00	1.00	1.03	1.00	1.00	1.00	1.03

Table A.4. Comparison of PaToH and Sanchis (SN) for RN model
SN1 and SN2 represent the Level 1 and Level 2 of Sanchis' algorithm. Minimum and Average show the minimum and average number of coupling rows after 20 runs. Time is the average running time of 20 runs in seconds. Columns of PaToH display the actual results, and other columns have been normalized with respect to PaToH.

Problem	k	Minimum			Average			Time		
		PaToH	SN-1	SN-2	PaToH	SN-1	SN-2	PaToH	SN-1	SN-2
25fv47	2	83	2.37	1.94	90.24	2.84	2.30	0.37	2.65	2.38
	4	116	3.11	3.33	139.81	3.82	3.38	0.67	2.04	1.87
	6	155	3.92	3.36	180.48	3.54	3.04	0.84	1.13	1.94
	8	189	3.46	2.97	220.76	3.02	2.62	0.94	1.37	2.20
80bau3b	2	74	4.20	3.65	91.52	4.06	3.55	1.19	16.32	13.09
	4	251	2.27	2.12	314.10	2.59	2.74	2.32	14.33	9.91
	6	343	4.07	3.58	390.14	3.89	3.28	3.03	2.88	3.81
	8	394	3.99	3.43	428.62	3.72	3.21	3.36	2.71	4.54
bnl2	2	121	2.69	2.26	143.71	2.56	2.35	0.60	4.50	3.83
	4	279	1.80	1.73	319.00	1.83	1.63	1.18	4.02	4.01
	6	350	1.98	1.52	395.62	2.01	1.56	1.52	3.24	3.99
	8	411	1.94	1.38	476.29	1.80	1.39	1.69	3.86	5.76
cycle	2	34	11.76	5.09	47.33	14.22	9.37	0.63	2.98	3.46
	4	94	8.26	8.20	135.86	7.26	6.30	1.24	2.26	3.10
	6	56	20.29	16.84	78.48	15.04	12.51	1.60	1.55	2.66
	8	193	6.25	5.12	225.43	5.48	4.53	1.82	1.77	3.20
czprob	2	176	0.28	0.19	221.81	0.34	0.18	0.80	1.91	1.30
	4	161	0.21	0.40	225.81	0.77	0.48	1.33	1.74	1.83
	6	152	2.22	1.86	230.71	1.62	1.31	1.68	1.52	1.14
	8	133	2.93	2.34	220.05	1.92	1.50	1.67	2.00	1.92
d2q06c	2	147	4.56	3.67	185.05	4.10	3.38	1.07	5.37	4.51
	4	251	4.29	3.82	282.24	5.02	3.99	2.23	3.00	4.65
	6	277	6.00	4.95	328.95	5.18	4.31	2.88	1.55	2.70
	8	315	5.55	4.67	369.52	4.81	4.03	3.31	1.56	3.03
ganges	2	28	3.79	3.29	37.71	3.46	3.46	0.28	2.21	2.29
	4	66	4.18	2.48	82.62	3.96	2.49	0.58	3.24	2.98
	6	85	5.56	2.46	105.90	5.03	2.34	0.74	3.03	3.97
	8	111	5.10	1.88	132.24	4.70	2.03	0.85	3.01	5.39
greenbea	2	91	6.16	5.43	108.24	6.00	5.50	1.13	4.18	3.01
	4	145	5.53	5.15	182.95	4.96	4.34	2.19	5.43	4.62
	6	192	6.86	4.76	245.81	5.68	3.89	2.90	2.64	4.44
	8	213	6.63	4.80	276.86	5.67	3.81	3.22	2.46	4.11

Table A.5. Comparison of PaToH and Sanchis (SN) for RN model (cont.d)

Problem	k	Min			Average			Time		
		PaToH	SN-1	SN-2	PaToH	SN-1	SN-2	PaToH	SN-1	SN-2
greenbeb	2	82	7.27	6.56	106.19	6.06	5.69	1.15	3.98	3.39
	4	147	5.46	5.08	194.48	4.86	4.07	2.24	4.31	4.34
	6	174	7.06	5.24	233.86	5.96	4.07	2.99	2.82	4.09
	8	236	6.13	4.33	276.33	5.68	3.78	3.26	2.38	4.51
scfxm3	2	10	12.70	5.40	13.19	15.91	11.65	0.24	2.46	2.58
	4	23	14.61	11.91	33.48	12.59	10.26	0.49	2.82	2.82
	6	42	13.29	9.45	51.05	12.14	8.80	0.66	1.61	2.64
	8	76	8.66	5.89	84.48	8.04	5.81	0.74	1.53	2.80
sctap2	2	41	2.12	1.34	46.14	2.55	2.28	0.23	3.78	3.78
	4	101	4.60	1.57	106.29	4.98	1.60	0.46	2.24	3.02
	6	116	4.67	1.52	133.90	4.42	1.40	0.60	4.35	4.97
	8	144	5.29	1.31	157.81	5.01	1.35	0.70	2.51	5.91
sctap3	2	40	2.80	1.98	50.05	2.90	2.70	0.28	5.82	4.57
	4	81	8.25	2.47	119.19	6.15	1.86	0.58	2.59	3.67
	6	135	5.41	1.72	152.38	5.24	1.69	0.79	5.11	5.46
	8	163	6.17	1.50	179.81	5.93	1.64	0.88	3.49	6.35
ship12l	2	10	19.00	19.00	10.29	19.43	18.74	0.62	4.27	2.42
	4	10	53.40	33.90	10.29	64.42	49.28	1.20	2.35	2.76
	6	10	78.90	72.60	10.14	80.94	74.85	1.61	1.04	2.00
	8	128	6.45	6.22	146.43	5.70	5.56	1.82	0.84	2.05
ship12s	2	10	18.40	17.70	10.00	19.40	18.82	0.27	3.00	2.52
	4	10	27.20	25.40	10.10	34.72	31.14	0.53	1.09	1.75
	6	10	35.10	35.00	10.00	36.70	35.73	0.69	0.93	1.39
	8	89	4.15	3.99	99.14	3.73	3.70	0.79	0.89	1.33
sierra	2	42	1.48	1.24	50.52	1.89	1.50	0.24	8.33	6.83
	4	81	5.99	3.21	95.10	5.41	3.10	0.46	2.17	2.85
	6	100	5.01	2.96	122.86	4.42	2.72	0.61	4.33	3.64
	8	121	4.98	2.83	139.14	4.64	2.77	0.67	6.63	4.78
stocfor2	2	11	4.91	4.45	19.71	8.04	5.61	0.33	3.12	3.39
	4	39	14.90	13.08	46.67	14.16	12.65	0.69	2.04	2.75
	6	104	6.72	6.23	116.81	6.20	5.86	0.91	1.96	2.51
	8	94	7.81	7.37	108.19	7.03	6.63	1.04	2.49	3.35
woodw	2	192	3.81	1.39	202.52	4.09	2.83	3.67	1.13	2.20
	4	432	2.49	2.43	465.81	2.33	2.28	6.70	0.21	0.41
	6	388	2.81	2.79	449.33	2.43	2.42	7.59	0.16	0.31
	8	447	2.44	2.44	491.14	2.23	2.22	7.74	0.26	0.44

Table A.6. Comparison of PaToh and Sanchis (SN) for RN model (cont.d)

Problem	k	Min			Average			Time		
		PaToH	SN-1	SN-2	PaToH	SN-1	SN-2	PaToH	SN-1	SN-2
cre-a	2	91	4.20	3.30	109.33	3.67	3.00	0.83	1.52	2.14
	4	124	3.90	3.48	142.62	3.52	3.13	1.38	2.57	2.03
	6	142	3.84	3.15	159.48	3.82	2.92	1.73	3.38	2.24
	8	145	4.50	3.17	165.57	4.32	2.86	1.88	4.30	3.11
cre-c	2	75	4.99	3.87	103.81	3.93	3.20	0.79	1.67	2.15
	4	109	4.45	3.92	129.05	3.99	3.50	1.27	2.75	2.03
	6	122	5.31	3.71	146.76	4.67	3.20	1.60	2.16	2.11
	8	127	5.50	3.63	150.76	4.86	3.20	1.75	4.07	3.19
cre-d	2	971	1.61	1.17	1238.14	1.89	1.86	21.57	40.94	56.95
	4	1204	4.78	4.56	1456.62	4.01	3.81	33.43	1.27	1.33
	6	1514	3.92	3.70	1682.81	3.57	3.39	39.90	1.90	2.82
	8	1511	4.04	3.88	1803.62	3.41	3.27	43.79	2.66	3.34
osa-07	2	1021	1.09	1.03	1036.71	1.07	1.05	10.43	0.26	0.77
	4	1013	1.10	1.10	1033.19	1.08	1.08	14.35	0.12	0.13
	6	1020	1.10	1.10	1039.52	1.08	1.08	16.78	0.12	0.14
	8	998	1.12	1.12	1034.26	1.08	1.08	16.49	0.19	0.23
CO9	2	782	0.96	0.82	1073.10	1.20	1.05	5.49	5.61	4.07
	4	912	2.56	1.78	1493.48	1.74	1.29	9.55	8.31	6.85
	6	1107	3.50	1.81	1601.05	2.50	1.38	11.74	5.47	7.52
	8	1216	3.55	1.75	1634.33	2.79	1.47	12.36	7.47	11.16
CQ9	2	668	1.52	1.34	988.19	1.24	1.10	4.99	5.55	4.04
	4	848	2.49	1.91	1243.71	1.85	1.43	8.61	7.68	5.81
	6	1005	3.47	1.88	1386.29	2.60	1.46	10.42	6.76	7.39
	8	854	4.87	2.37	1454.81	2.91	1.57	10.96	7.61	10.73
GE	2	208	6.26	5.69	236.52	6.17	5.53	1.93	7.84	7.72
	4	332	6.69	5.71	409.67	6.09	5.16	3.71	10.76	11.06
	6	527	6.18	4.58	577.29	6.06	4.34	4.86	12.54	10.91
	8	596	6.72	4.51	643.67	6.55	4.30	5.34	11.38	14.34
NL	2	360	1.49	1.33	456.62	1.46	1.32	2.23	5.30	4.72
	4	467	3.10	1.96	607.29	2.57	1.67	3.75	8.33	7.26
	6	633	3.94	1.65	703.38	3.66	1.60	4.64	9.07	6.69
	8	644	4.65	1.63	749.05	4.22	1.57	4.96	9.80	9.34
mod2	2	354	7.62	5.44	393.00	9.27	7.62	8.24	13.16	9.39
	4	678	10.85	9.23	786.90	9.61	8.24	15.20	8.80	10.74
	6	891	10.68	7.71	1067.19	9.08	6.59	19.86	9.79	11.70
	8	1310	8.13	5.48	1461.76	7.45	4.99	21.67	11.83	15.12
world	2	370	7.30	6.58	407.05	8.43	7.65	8.47	13.73	9.68
	4	707	9.56	8.61	797.05	8.96	7.87	15.62	12.68	12.95
	6	927	8.67	7.05	1091.76	7.66	6.29	20.49	11.61	12.57
	8	1316	7.21	5.33	1487.43	6.56	4.85	20.49	15.07	18.58

Table A.7. General Comparison of PaToH and Sanchis on RN Model
All values have been normalized with respect to PaToH.

k	Min			Average			Time		
	PaToH	SN1	SN2	PaToH	SN1	SN2	PaToH	SN1	SN2
2	1.00	5.38	4.26	1.00	5.78	4.94	1.00	6.36	6.19
4	1.00	8.00	6.24	1.00	8.27	6.62	1.00	4.41	4.35
6	1.00	9.65	7.90	1.00	9.08	7.48	1.00	3.80	4.29
8	1.00	5.12	3.53	1.00	4.57	3.18	1.00	4.23	5.59
Avg	1.00	7.04	5.48	1.00	6.92	5.55	1.00	4.70	5.10

Table A.8. Comparison of PaToH and Sanchis (SN) for CN model
SN1 and SN2 represent the Level 1 and Level 2 of Sanchis' algorithm. Columns of PaToH display the actual results, and other columns have been normalized with respect to PaToH.

Problem	k	Min			Average			Time		
		PaToH	SN-1	SN-2	PaToH	SN-1	SN-2	PaToH	SN-1	SN-2
25fv47	2	155	1.16	1.04	192.71	1.41	1.28	0.39	1.67	4.62
	4	310	1.87	1.52	341.05	2.08	1.82	0.73	2.03	7.49
	6	338	2.73	2.05	367.00	2.87	2.38	0.92	1.41	52.57
	8	475	2.23	1.79	520.19	2.24	1.80	1.06	1.43	138.56
80bau3b	2	195	2.11	2.46	217.14	3.75	3.77	1.06	1.35	1.48
	4	612	1.98	1.72	704.67	2.02	1.93	2.15	1.31	2.01
	6	853	1.77	1.76	984.57	1.85	1.67	2.86	1.50	2.53
	8	1068	1.76	1.57	1179.14	1.76	1.56	3.04	2.26	4.06
bnl2	2	108	2.53	2.06	130.38	3.17	3.08	0.78	1.51	1.94
	4	272	2.71	1.76	339.71	2.86	2.06	1.46	1.68	2.34
	6	369	3.25	1.39	430.86	3.06	1.89	1.96	2.37	3.69
	8	448	3.07	1.63	526.90	3.07	1.74	2.05	3.04	5.06
cycle	2	110	3.55	2.43	130.14	4.36	4.35	0.63	2.60	2.98
	4	195	4.82	4.73	218.48	6.17	4.96	1.27	2.13	3.26
	6	217	6.79	6.14	256.90	6.88	5.81	1.72	1.53	2.52
	8	239	6.92	5.74	316.67	6.02	5.10	1.92	1.46	2.89
d2q06c	2	339	2.63	2.59	418.33	2.85	2.76	1.10	1.76	1.93
	4	662	2.54	2.76	712.81	3.02	2.64	2.09	2.01	2.57
	6	753	3.34	2.65	848.90	3.10	2.63	2.79	1.65	3.21
	8	875	3.01	2.70	937.57	2.97	2.67	3.05	1.79	3.62
ganges	2	40	2.17	1.65	55.19	4.18	2.73	0.24	2.33	2.96
	4	103	4.72	4.64	125.95	4.17	4.17	0.50	1.74	2.44
	6	124	4.25	4.36	170.10	3.38	3.42	0.67	2.01	2.90
	8	196	2.98	2.98	223.62	2.75	2.74	0.73	2.36	4.27
greenbea	2	279	2.63	2.11	337.43	2.88	3.26	1.45	1.68	2.47
	4	483	5.68	4.35	580.95	5.36	4.10	2.69	0.91	2.79
	6	632	5.12	4.10	719.05	4.92	3.79	3.47	1.31	3.51
	8	805	4.70	3.56	947.48	4.28	3.22	3.85	1.09	17.89
greenbeb	2	281	2.11	2.57	340.90	2.97	3.61	1.45	1.70	2.32
	4	488	5.28	3.99	577.19	5.33	4.07	2.76	0.96	2.35
	6	620	4.66	3.83	726.38	4.82	3.70	3.60	1.28	4.29
	8	780	4.97	3.68	884.24	4.55	3.24	3.96	1.05	18.71
scfxm3	2	23	3.43	3.52	30.43	4.75	4.75	0.29	1.59	2.07
	4	59	5.07	3.97	71.57	6.34	4.62	0.58	1.79	2.83
	6	83	7.83	3.87	98.29	7.32	4.81	0.72	1.14	2.96
	8	159	4.29	2.74	181.52	4.08	2.92	0.86	1.72	3.78
sctap2	2	131	2.06	1.78	165.71	2.20	1.86	0.27	1.11	1.70
	4	279	2.14	1.95	294.76	2.30	2.09	0.55	1.27	1.69
	6	363	2.24	1.75	398.10	2.21	1.79	0.70	1.77	2.34
	8	446	2.07	1.56	463.76	2.19	1.68	0.77	2.22	3.34
sctap3	2	186	1.54	1.53	205.86	2.13	2.01	0.40	1.25	1.70
	4	347	2.44	1.73	370.14	2.75	2.08	0.77	1.03	1.78
	6	466	2.28	1.79	507.81	2.27	1.79	0.99	1.97	2.47
	8	523	2.53	1.71	551.71	2.60	1.75	1.10	1.62	3.36

Table A.9. Comparison of PaToH and Sanchis (SN) for CN model (cont.d)

Problem	k	Min			Average			Time		
		PaToH	SN-1	SN-2	PaToH	SN-1	SN-2	PaToH	SN-1	SN-2
ship12s	2	59	1.29	1.03	67.67	3.22	3.29	0.51	0.84	1.37
	4	117	2.09	2.13	154.38	3.08	3.08	1.04	0.80	1.48
	6	127	2.22	2.25	205.76	2.67	2.68	1.41	0.96	1.55
	8	232	1.60	1.29	268.86	2.25	2.12	1.59	1.15	2.22
sierra	2	58	1.07	1.03	68.90	1.87	1.81	0.37	1.27	1.70
	4	188	1.48	1.22	201.76	2.48	1.60	0.75	1.71	2.84
	6	254	2.47	1.53	288.86	3.27	1.82	0.99	2.16	2.98
	8	290	3.75	1.52	340.71	3.61	1.99	1.07	2.52	3.93
stocfor2	2	13	8.08	8.31	29.24	7.96	4.76	0.35	2.57	2.91
	4	39	14.69	9.08	65.33	9.97	6.56	0.68	3.94	3.87
	6	91	8.98	5.48	106.62	8.46	5.88	0.91	2.25	4.89
	8	90	9.96	7.17	118.90	7.98	6.34	1.00	2.87	6.04
cre-a	2	610	1.18	1.12	658.10	1.25	1.24	1.06	1.65	2.62
	4	983	1.39	1.22	1043.10	1.44	1.25	2.03	1.48	4.33
	6	1158	1.53	1.19	1192.24	1.57	1.23	2.53	1.88	10.23
	8	1231	1.56	1.20	1268.95	1.62	1.25	2.87	3.00	28.42
cre-c	2	542	1.25	1.16	596.71	1.23	1.22	0.90	1.41	2.69
	4	890	1.27	1.15	932.67	1.36	1.20	1.78	1.37	4.02
	6	1012	1.34	1.15	1056.00	1.40	1.19	2.23	2.00	8.90
	8	1087	1.43	1.16	1126.71	1.49	1.18	2.63	2.26	13.14
cre-d	2	6828	0.78	1.04	7302.81	0.92	0.97	40.40	0.41	0.72
	4	22113	1.11	—	22923.29	1.27	—	57.92	0.58	—
	6	28851	1.17	—	30265.47	1.20	—	69.64	0.77	
	8	33045	1.11	—	34141.43	1.18	—	70.93	1.02	—
CO9	2	1087	2.17	1.61	1432.76	2.34	1.88	5.71	2.64	2.76
	4	2247	2.27	2.17	2426.05	2.49	2.24	10.52	4.04	4.36
	6	2712	2.61	2.10	2976.05	2.59	2.11	13.33	3.16	10.16
	8	2870	2.82	—	3164.33	2.75	—	14.44	2.74	—
CQ9	2	1064	2.22	2.14	1427.43	2.01	1.91	4.96	2.71	2.20
	4	2065	2.38	1.99	2191.33	2.54	2.15	9.33	3.22	3.45
	6	2473	2.76	2.19	2665.52	2.56	2.14	11.53	3.73	12.84
	8	2516	—	—	2815.62	—	—	12.58	—	—
GE	2	313	3.78	3.12	368.38	3.75	3.88	1.95	7.14	6.60
	4	588	4.99	4.15	663.38	4.61	4.00	3.79	11.08	8.72
	6	791	4.60	3.56	900.95	4.23	3.44	5.05	10.46	19.06
	8	1003	4.22	—	1076.67	4.08	—	5.50	11.77	—
NL	2	1019	1.26	1.25	1127.71	1.48	1.43	2.43	2.96	3.14
	4	1657	1.71	1.43	1812.71	1.67	1.41	4.28	5.03	4.59
	6	1857	1.84	1.43	2013.81	1.85	1.41	5.35	5.12	9.85
	8	1969	2.05	—	2207.48	1.92	—	5.76	5.44	—
mod2	2	2011	2.16	2.28	2097.33	2.48	2.53	10.31	11.96	12.33
	4	2990	3.22	2.65	3224.14	3.61	2.78	20.92	21.26	23.95
	6	3391	5.19	3.23	3635.29	5.23	3.27	27.47	11.92	42.91
	8	3926	5.32	—	4093.71	5.32	—	30.56	10.41	—
world	2	2389	1.40	2.06	2463.14	2.21	2.23	10.82	12.05	11.59
	4	3645	2.45	2.27	3775.24	3.04	2.42	21.98	21.93	22.30
	6	4094	5.14	2.87	4248.00	5.03	2.97	28.62	4.47	29.52
	8	4583	4.63	—	4723.71	4.69	—	31.75	9.38	—

Table A.10. Results of Column-Net with transfer model with PaToh Min , Max and Avg fields denote the minimum, maximum , and average number of coupling rows in the block angular matrix after 20 runs. Columns 4 and 5 display percentage of coupling of to the original matrix. Column 6 display the difference between the average and minimum results. Time field display the partitioning time in seconds.

Problem	k	Min	Max	Avg	$\frac{min}{nets} * 100$	$\frac{avg}{nets} * 100$	$\frac{avg-min}{min}$	time
25fv47	2	158	239	186.67	19.24	22.74	18.14	0.40
	4	321	434	387.52	39.10	47.20	20.72	0.78
	6	379	473	422.14	46.16	51.42	11.38	1.00
	8	589	685	629.57	71.74	76.68	6.89	1.15
80bau3b	2	200	233	215.52	8.84	9.53	7.76	1.06
	4	606	928	746.52	26.79	33.00	23.19	2.16
	6	855	1181	1038.10	37.80	45.89	21.41	2.86
	8	1153	1309	1231.62	50.97	54.45	6.82	3.15
bnl2	2	109	275	141.33	4.69	6.08	29.66	0.77
	4	271	396	333.05	11.66	14.33	22.90	1.50
	6	385	499	432.81	16.57	18.62	12.42	2.00
	8	598	728	643.76	25.73	27.70	7.65	2.12
cycle	2	115	173	130.76	6.04	6.87	13.71	0.65
	4	178	273	227.95	9.35	11.98	28.06	1.33
	6	217	362	260.57	11.40	13.69	20.08	1.75
	8	239	348	307.14	12.56	16.14	28.51	1.99
czprob	2	744	1037	849.10	80.09	91.40	14.13	2.83
	4	1619	1892	1715.29	174.27	184.64	5.95	4.87
	6	1936	2208	2056.29	208.40	221.34	6.21	6.23
	8	2182	2431	2272.52	234.88	244.62	4.15	6.47
d2q06c	2	326	497	409.52	15.02	18.86	25.62	1.12
	4	601	801	711.57	27.68	32.78	18.40	2.22
	6	798	991	895.00	36.76	41.23	12.16	2.89
	8	927	1143	987.90	42.70	45.50	6.57	3.27
ganges	2	36	93	53.52	2.75	4.09	48.68	0.25
	4	104	161	125.43	7.94	9.58	20.60	0.51
	6	145	235	185.43	11.08	14.17	27.88	0.71
	8	190	250	229.10	14.51	17.50	20.58	0.75
greenbea	2	274	387	322.52	11.45	13.48	17.71	1.45
	4	531	763	624.52	22.20	26.11	17.61	2.77
	6	699	973	803.33	29.22	33.58	14.93	3.67
	8	901	1354	1042.52	37.67	43.58	15.71	4.06

Table A.11. Results of Column-Net with transfer model with PaToh (cont.d)

Problem	k	Min	Max	Avg	$\frac{min}{nets} * 100$	$\frac{avg}{nets} * 100$	$\frac{avg-min}{min}$	time
greenbeb	2	274	392	328.05	11.45	13.71	19.73	1.44
	4	521	698	627.67	21.78	26.24	20.47	2.75
	6	675	950	798.33	28.22	33.38	18.27	3.67
	8	867	1224	1056.29	36.25	44.16	21.83	4.10
scfxm3	2	24	47	31.33	2.42	3.16	30.56	0.28
	4	59	107	78.67	5.96	7.95	33.33	0.59
	6	84	131	100.19	8.48	10.12	19.27	0.72
	8	167	219	189.81	16.87	19.17	13.66	0.88
sctap2	2	131	195	166.19	12.02	15.25	26.86	0.27
	4	291	373	328.71	26.70	30.16	12.96	0.54
	6	382	444	411.81	35.05	37.78	7.80	0.73
	8	478	568	512.48	43.85	47.02	7.21	0.80
sctap3	2	185	227	203.00	12.50	13.72	9.73	0.39
	4	362	460	400.76	24.46	27.08	10.71	0.79
	6	490	596	522.95	33.11	35.33	6.72	1.01
	8	560	631	599.57	37.84	40.51	7.07	1.17
ship12l	2	14	208	72.29	1.22	6.28	416.33	0.97
	4	19	346	96.33	1.65	8.37	407.02	1.89
	6	21	664	251.38	1.82	21.84	1097.05	2.49
	8	459	860	602.71	39.88	52.36	31.31	2.88
ship12s	2	59	72	66.43	5.13	5.77	12.59	0.50
	4	118	218	144.33	10.25	12.54	22.32	1.07
	6	127	325	210.38	11.03	18.28	65.65	1.43
	8	233	313	268.10	20.24	23.29	15.06	1.61
sierra	2	51	93	65.38	4.16	5.33	28.20	0.38
	4	192	235	202.67	15.65	16.52	5.56	0.74
	6	236	328	295.95	19.23	24.12	25.40	0.95
	8	331	407	346.67	26.98	28.25	4.73	1.07
stocfor2	2	13	51	27.33	0.60	1.27	110.26	0.34
	4	43	103	67.86	1.99	3.15	57.81	0.69
	6	89	126	103.05	4.13	4.78	15.78	0.92
	8	103	181	136.24	4.78	6.32	32.27	1.01
woodw	2	935	1305	1052.95	85.15	95.90	12.62	1.73
	4	2493	3525	3248.05	227.05	295.81	30.29	3.28
	6	3303	5980	4508.81	300.82	410.64	36.51	4.29
	8	4676	6022	5261.52	425.87	479.19	12.52	4.59

Table A.12. Results of Column-Net with transfer model with PaToh (cont'd)

Problem	k	Min	Max	Avg	$\frac{min}{nets} * 100$	$\frac{avg}{nets} * 100$	$\frac{avg-min}{min}$	time
cre-a	2	587	716	644.62	16.70	18.33	9.82	1.05
	4	1033	1201	1107.90	29.38	31.51	7.25	2.02
	6	1277	1379	1324.86	36.32	37.68	3.75	2.64
	8	1404	1537	1463.24	39.93	41.62	4.22	2.93
cre-c	2	548	647	591.29	17.86	19.27	7.90	0.88
	4	970	1065	1023.33	31.62	33.36	5.50	1.70
	6	1160	1304	1222.86	37.81	39.86	5.42	2.26
	8	1261	1408	1314.24	41.10	42.84	4.22	2.47
cre-d	2	6620	7566	7262.53	74.17	81.36	9.71	41.93
	4	22899	24811	23856.40	256.54	267.27	4.18	59.42
	6	30945	32966	31772.93	346.68	355.96	2.68	79.24
	8	36857	39738	38417.93	412.92	430.40	4.24	76.79
osa-07	2	10353	15764	13817.24	926.03	1235.89	33.46	26.69
	4	19630	29096	24215.95	1755.81	2166.01	23.36	41.15
	6	23275	32177	28791.05	2081.84	2575.23	23.70	49.00
	8	27933	39518	33724.33	2498.48	3016.49	20.73	49.80
CO9	2	1093	1781	1501.62	10.13	13.92	37.39	5.72
	4	2460	2921	2609.95	22.80	24.19	6.10	11.10
	6	2959	3496	3173.81	27.43	29.42	7.26	14.27
	8	3197	3843	3540.62	29.63	32.82	10.75	15.70
CQ9	2	1025	1549	1322.37	11.05	14.25	29.01	4.79
	4	2037	2496	2250.21	21.96	24.25	10.47	9.09
	6	2448	3085	2619.32	26.38	28.23	7.00	12.20
	8	2625	3349	2921.26	28.29	31.49	11.29	12.73
GE	2	318	456	372.33	3.15	3.69	17.09	1.96
	4	556	783	650.29	5.51	6.44	16.96	3.82
	6	816	1008	922.48	8.08	9.13	13.05	5.12
	8	1035	1230	1117.81	10.25	11.07	8.00	5.51
NL	2	1023	1267	1110.05	14.53	15.77	8.51	2.45
	4	1798	2141	1984.29	25.54	28.19	10.36	4.63
	6	2180	2662	2362.00	30.97	33.56	8.35	6.06
	8	2410	2838	2613.38	34.24	37.13	8.44	6.50
mod2	2	2014	2256	2086.67	5.79	6.00	3.61	10.33
	4	3266	3440	3348.57	9.39	9.63	2.53	19.59
	6	3756	4091	3877.67	10.80	11.15	3.24	25.73
	8	4385	4893	4548.05	12.61	13.08	3.72	27.87
world	2	2357	2551	2440.19	6.83	7.07	3.53	10.77
	4	3733	4060	3912.43	10.82	11.34	4.81	20.34
	6	4361	4681	4508.24	12.64	13.07	3.38	27.12
	8	5197	5394	5293.19	15.06	15.34	1.85	29.22

Table A.13. Comparisons of Greedy Heuristics for RIG
MI, MR, OM and MF denote the maximum inclusion, minimum removal , one-max and maximum flow solutions. Minimum and Average are minimum and average number of coupling rows after 20 runs for 8-way partitions. All values have been normalized with respect to OM. F_{time} shows the run time of maximum flow soltuion in seconds.

Problem	Minimum				Average				F_{time}
	OM	MI	MR	MF	OM	MI	MR	MF	
25fv47	1.000	1.011	1.022	1.017	1.000	1.015	1.026	1.024	11.81
80bau3b	1.000	1.045	1.018	1.050	1.000	1.037	1.016	1.038	59.05
bnl2	1.000	1.034	1.005	1.007	1.000	1.036	0.997	0.999	10.17
cycle	1.000	1.000	1.000	1.000	1.000	1.002	0.999	1.001	2.05
czprob	1.000	1.000	1.000	3.103	1.000	1.000	1.000	3.508	141.53
d2q06c	1.000	1.000	0.997	1.007	1.000	1.006	1.001	1.015	37.49
ganges	1.000	1.000	1.000	1.000	1.000	1.004	1.000	1.001	1.68
greenbea	1.000	1.008	1.004	1.000	1.000	1.038	1.011	1.021	24.40
greenbeb	1.000	1.017	1.000	1.009	1.000	1.044	1.005	1.013	24.25
scfxm3	1.000	1.039	1.000	1.000	1.000	1.038	1.011	1.014	1.14
sctap2	1.000	1.006	1.023	1.000	1.000	1.006	1.031	1.001	7.15
sctap3	1.000	1.000	1.018	0.991	1.000	1.005	1.018	0.994	9.95
ship12l	1.000	1.000	1.013	1.063	1.000	1.000	1.014	1.058	3.40
ship12s	1.000	1.000	1.014	1.081	1.000	1.001	1.007	1.290	4.97
sierra	1.000	1.000	1.014	1.029	1.000	1.012	0.999	1.021	3.79
woodw	1.000	1.008	1.004	1.078	1.000	1.010	1.013	1.076	1059.82
cre-a	1.000	1.014	1.014	—	1.000	1.009	1.002	—	—
cre-c	1.000	1.004	1.000	—	1.000	1.007	1.001	—	—
cre-d	1.000	1.028	1.030	—	1.000	1.026	1.021	—	—
osa-07	1.000	1.000	1.000	—	1.000	1.000	1.009	—	—
CO9	1.000	1.015	1.006	—	1.000	1.018	1.008	—	—
CQ9	1.000	1.024	1.020	—	1.000	1.020	1.020	—	—
GE	1.000	1.012	1.000	—	1.000	1.009	1.000	—	—
NL	1.000	1.017	1.008	—	1.000	1.020	1.009	—	—
mod2	1.000	1.048	1.000	—	1.000	1.048	1.003	—	—
world	1.000	1.065	0.989	—	1.000	1.040	0.999	—	—
Avg	1.000	1.011	1.008	1.152	1.000	1.023	1.010	1.187	

Table A.14. Comparison of Greedy heuristics with Optimal solutions

M represent the optimal solution with matching, MI, MR, and OM represent the maximum inclusion, minimum recover, and One-Max heuristics, respectively. Minimum and Average columns represent the minimum and average seperator sizes for different methods when they start from the same wide separator. *Imp%* is the average of $((Min(MI, MR, OM) - Match) * 100)/Match$ values for 20 runs.

Problem	Minimum				Average				*Imp%*
	M	MI	MR	OM	M	MI	MR	OM	
25fv47	1.00	1.00	1.01	1.00	1.00	1.01	1.01	1.00	0.10
80bau3b	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	0.00
bnl2	1.00	1.08	1.00	1.00	1.00	1.09	1.00	1.00	0.12
cycle	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
czprob	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	0.00
d2q06c	1.00	1.02	1.00	1.01	1.00	1.02	1.00	1.00	0.03
ganges	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
greenbea	1.00	1.02	1.00	1.00	1.00	1.01	1.00	1.00	0.04
greenbeb	1.00	1.02	1.01	1.00	1.00	1.02	1.00	1.00	0.00
scfxm3	1.00	1.00	1.00	1.00	1.00	1.02	1.00	1.00	0.00
sctap2	1.00	1.07	1.07	1.01	1.00	1.02	1.03	1.00	0.48
sctap3	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	0.31
ship12l	1.00	1.00	1.00	1.00	1.00	1.00	1.01	1.00	0.00
ship12s	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
sierra	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00
stocfor2	1.00	1.06	1.00	1.00	1.00	1.05	1.00	1.00	0.00
woodw	1.00	1.00	1.16	1.00	1.00	1.01	1.14	1.00	0.32
cre-a	1.00	1.00	1.01	1.00	1.00	1.01	1.00	1.00	0.00
cre-c	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	0.00
cre-d	1.00	1.01	1.01	1.00	1.00	1.03	1.02	1.01	0.58
osa-07	1.00	1.00	1.07	1.00	1.00	1.00	1.15	1.00	0.00
CO9	1.00	1.02	1.00	1.00	1.00	1.03	1.01	1.00	0.17
CQ9	1.00	1.03	1.00	1.01	1.00	1.02	1.03	1.00	0.22
GE	1.00	1.00	1.00	1.00	1.00	1.01	1.00	1.00	0.00
NL	1.00	1.01	1.00	1.01	1.00	1.02	1.01	1.00	0.18
mod2	1.00	1.03	1.00	1.00	1.00	1.04	1.01	1.01	0.46
world	1.00	1.01	1.01	1.00	1.00	1.04	1.01	1.00	0.20
Avg	1.00	1.01	1.01	1.00	1.00	1.02	1.02	1.00	0.12

Table A.15. Comparison of Wide Separators for different methods

RIG represents finding wide separator by minimizing the edges on the cut. W1–W6 are weightening schems explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. RIG column present the actual value of average wide separator sizes after 20 runs. Other columns have been normalized with respect to this one.

Problem	RIG	W1	W2	W3	W4	W5	W6	Hy
25fv47	623.43	0.993	1.003	0.996	1.000	1.009	1.009	0.958
80bau3b	1067.71	0.901	0.955	0.949	0.934	0.932	0.962	0.757
bnl2	1186.52	0.956	0.973	0.970	0.952	0.952	0.966	0.855
cycle	407.71	0.995	0.994	1.034	1.073	1.111	1.020	1.331
czprob	798.19	1.002	0.996	1.002	0.996	1.012	1.004	0.818
d2q06c	1033.76	1.038	1.011	1.056	1.030	1.044	1.036	0.945
ganges	416.38	0.994	0.979	1.003	1.000	0.911	0.923	0.767
greenbea	1079.10	1.017	1.016	1.034	1.026	1.044	1.019	0.866
greenbeb	1060.14	1.030	1.038	1.032	1.047	1.074	1.032	0.872
scfxm3	279.00	1.017	1.012	1.051	1.072	1.035	1.033	0.933
sctap2	623.29	0.999	1.001	1.004	1.003	1.013	0.998	0.967
sctap3	754.86	1.019	0.997	1.014	1.026	1.029	1.044	0.974
ship04l	288.00	0.968	0.979	1.043	1.065	1.099	1.096	0.779
ship04s	229.43	0.995	1.087	1.151	1.188	1.134	1.090	0.960
ship08l	164.71	0.947	0.943	0.956	0.949	0.875	0.936	0.642
ship08s	251.10	0.959	0.912	0.899	0.906	0.999	0.918	0.291
ship12l	268.00	1.008	1.030	0.998	0.971	0.988	0.988	0.910
ship12s	476.48	0.994	1.031	1.051	1.006	1.056	0.997	0.401
sierra	424.67	1.012	1.019	1.034	1.026	1.013	1.005	0.990
stocfor2	373.71	0.976	1.009	0.957	1.008	0.979	1.018	0.694
woodw	1069.52	0.984	0.971	0.962	0.965	0.980	0.990	0.883
cre-a	3001.67	1.003	1.013	1.013	1.013	1.016	1.019	0.847
cre-c	2647.24	0.994	1.002	1.000	0.998	1.000	0.997	0.852
cre-d	5983.48	0.995	0.989	0.995	0.995	0.992	1.000	0.858
osa-07	1118.00	1.000	1.000	1.000	1.000	1.000	1.000	1.000
CO9	5695.81	0.981	0.981	1.021	1.060	1.071	1.067	0.798
CQ9	4662.14	0.952	0.949	0.965	1.029	1.112	1.042	0.817
GE	1881.05	0.983	0.992	0.985	0.994	0.969	1.012	0.863
NL	4735.05	0.938	0.891	0.882	0.881	0.894	0.884	0.733
mod2	11317.81	0.987	1.002	1.008	1.013	1.022	0.999	0.884
world	11686.71	1.000	1.010	1.013	1.017	1.035	1.013	0.892
Avg	1.000	0.991	0.995	1.001	1.005	1.011	1.003	0.869

Table A.16. Comparison of Edge Cuts for different methods

RIG represents finding wide separator by minimizing the edges on the cut. W1–W6 are weightening schems explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. RIG column present the actual value of average number of edges on cut sizes after 20 runs. Other columns have been normalized with respect to this one.

Problem	RIG	W1	W2	W3	W4	W5	W6	Hy
25fv47	2179.19	1.025	1.032	1.029	1.052	1.057	1.037	1.846
80bau3b	1440.62	1.089	1.046	1.018	1.060	1.051	1.006	1.669
bnl2	1665.24	1.027	1.022	1.012	1.014	1.027	0.994	2.241
cycle	461.86	0.997	1.093	1.236	1.390	1.551	1.222	6.704
czprob	4098.67	1.002	1.000	1.001	1.004	1.005	1.003	1.338
d2q06c	1798.67	1.082	1.069	1.184	1.166	1.218	1.127	2.353
ganges	288.05	1.024	1.073	1.388	1.587	1.992	1.716	2.716
greenbea	4053.00	1.042	1.050	1.106	1.117	1.172	1.112	1.753
greenbeb	4017.76	1.048	1.087	1.105	1.133	1.176	1.112	1.708
scfxm3	379.71	1.136	1.209	1.278	1.339	1.266	1.168	1.999
sctap2	1181.67	0.993	0.995	1.000	1.004	1.012	0.992	1.411
sctap3	1391.33	1.027	1.002	1.038	1.038	1.045	1.064	1.470
ship12l	812.19	1.008	1.111	1.118	1.068	1.104	1.125	2.456
ship12s	918.48	1.000	1.027	1.051	1.009	1.069	0.995	1.119
sierra	542.71	1.019	1.060	1.036	1.039	1.034	1.004	1.509
stocfor2	379.38	1.004	1.058	1.008	1.041	1.000	1.063	2.293
woodw	8793.76	1.006	1.008	1.018	1.024	1.027	1.029	1.293
cre-a	6472.10	1.026	1.045	1.054	1.063	1.074	1.060	1.635
cre-c	6287.43	1.029	1.038	1.039	1.045	1.051	1.035	1.573
cre-d	67476.48	0.992	0.979	0.984	0.998	0.994	1.026	1.583
osa-07	42845.90	1.000	1.000	1.000	1.008	1.008	1.008	0.979
CO9	15078.52	1.003	1.078	1.298	1.439	1.573	1.485	1.533
CQ9	14655.71	1.012	1.140	1.233	1.366	1.611	1.420	1.442
GE	1745.62	1.017	1.015	1.008	1.018	1.015	1.033	1.312
NL	8701.81	1.094	1.436	1.633	1.847	2.099	1.813	1.926
mod2	19607.48	0.998	1.042	1.075	1.090	1.112	1.060	1.951
world	20671.19	1.025	1.037	1.071	1.088	1.144	1.064	1.854
Avg	1.000	1.027	1.065	1.112	1.150	1.203	1.140	1.914

Table A.17. Comparison of Minimum Separator Sizes for different methods

RIG represents finding wide separator by minimizing the edges on the cut. W1–W6 are weightening schemes explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. RIG column present the actual average value of minimum number of coupling rows after 20 runs. Other columns have been normalized with respect to this one.

Problem	RIG	W1	W2	W3	W4	W5	W6	Hy
25fv47	181	0.994	0.923	0.956	0.945	0.906	0.923	1.309
80bau3b	337	0.843	0.908	0.935	0.887	0.944	0.979	0.653
bnl2	408	1.002	1.027	1.029	0.973	0.961	1.042	1.083
cycle	96	1.010	1.021	1.042	1.021	1.042	1.031	1.062
czprob	29	1.000	1.000	1.000	1.000	1.000	1.000	0.931
d2q06c	296	1.007	0.976	1.003	0.997	0.990	1.010	1.135
ganges	150	1.020	0.913	0.947	0.900	0.853	0.827	0.767
greenbea	237	0.966	0.983	1.013	0.958	0.975	1.000	1.063
greenbeb	232	0.927	0.987	1.009	1.009	1.026	0.996	1.073
scfxm3	77	0.948	0.961	0.935	0.974	0.935	0.870	0.974
sctap2	177	0.977	0.994	1.006	0.989	0.994	0.994	1.418
sctap3	217	0.903	0.945	0.917	0.912	0.871	0.963	1.336
ship12l	79	0.924	0.962	0.949	0.975	0.987	0.975	0.886
ship12s	74	0.986	0.932	0.932	0.905	0.946	0.851	0.689
sierra	139	1.079	1.050	1.029	1.000	0.964	1.000	1.266
stocfor2	104	1.115	1.106	1.048	1.144	1.154	0.933	1.038
woodw	245	0.959	0.686	0.784	0.751	0.653	0.629	1.196
cre-a	214	0.799	0.724	0.673	0.654	0.659	0.692	2.126
cre-c	238	0.761	0.634	0.576	0.492	0.534	0.546	2.008
cre-d	1293	0.964	0.927	0.863	0.802	0.864	0.776	1.295
osa-07	80	1.013	1.000	1.000	1.000	1.000	1.000	0.962
CO9	2360	0.856	0.736	0.690	0.639	0.578	0.592	0.770
CQ9	1857	0.865	0.755	0.699	0.651	0.571	0.622	0.828
GE	563	0.950	0.968	0.977	0.989	0.918	0.966	0.897
NL	1914	0.706	0.546	0.479	0.413	0.331	0.451	0.624
mod2	1314	0.888	0.799	0.801	0.785	0.783	0.783	2.002
world	1297	0.951	0.854	0.862	0.837	0.828	0.892	2.147
Avg	1.000	0.941	0.901	0.895	0.874	0.862	0.865	1.168

Table A.18. Comparison of Average Separator Sizes for different methods

RIG represents finding wide separator by minimizing the edges on the cut. W1–W6 are weightening schems explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. RIG column present the actual value of average number of coupling rows after 20 runs. Other columns have been normalized with respect to this one.

Problem	RIG	W1	W2	W3	W4	W5	W6	Hy
25fv47	188.52	1.023	1.000	0.984	0.994	1.002	0.987	1.399
80bau3b	357.90	0.856	0.950	0.973	0.946	0.956	0.985	0.722
bnl2	456.00	0.962	0.968	0.974	0.952	0.932	0.981	1.067
cycle	109.90	1.011	0.992	1.049	1.131	1.162	1.053	1.537
czprob	31.67	0.985	0.946	0.926	0.923	0.917	0.928	0.928
d2q06c	310.76	1.007	1.010	1.033	1.028	1.035	1.017	1.234
ganges	174.62	1.003	0.976	0.996	1.016	0.899	0.918	0.756
greenbea	263.05	1.016	1.011	1.035	1.015	1.051	1.000	1.135
greenbeb	254.24	1.026	1.048	1.033	1.062	1.100	1.059	1.147
scfxm3	83.90	0.982	0.974	0.965	1.014	0.934	0.975	1.144
sctap2	187.38	0.976	0.982	0.986	0.988	0.980	0.980	1.446
sctap3	225.57	0.940	0.980	0.948	0.934	0.920	0.963	1.441
ship12l	86.90	0.988	0.999	1.007	0.965	1.012	0.994	0.958
ship12s	85.05	0.974	1.010	1.025	0.978	1.016	0.969	0.900
sierra	157.90	1.018	1.026	1.013	1.011	0.921	0.994	1.239
stocfor2	138.52	0.958	0.978	0.941	0.981	0.969	0.994	0.937
woodw	292.29	0.957	0.744	0.741	0.700	0.675	0.677	1.148
cre-a	251.38	0.779	0.696	0.654	0.647	0.615	0.649	2.055
cre-c	284.67	0.727	0.601	0.578	0.542	0.525	0.564	1.830
cre-d	1442.71	0.942	0.908	0.852	0.834	0.861	0.787	1.285
osa-07	80.90	1.001	0.994	0.993	0.992	0.994	0.989	0.975
CO9	2430.19	0.893	0.798	0.726	0.676	0.615	0.654	0.840
CQ9	2041.67	0.846	0.717	0.667	0.637	0.570	0.619	0.830
GE	610.90	0.982	0.987	0.976	0.977	0.943	0.988	0.928
NL	2046.62	0.821	0.644	0.557	0.487	0.396	0.514	0.704
mod2	1517.71	0.863	0.837	0.823	0.823	0.790	0.805	2.124
world	1598.10	0.878	0.834	0.788	0.770	0.758	0.798	2.010
Avg	1.000	0.941	0.911	0.898	0.890	0.872	0.883	1.212

Table A.19. Comparison of run times for different methods

RIG represents finding wide separator by minimizing the edges on the cut. W1 - W6 are weightening schems explained in Section 6.6.2. Hy is the hypergraph model of Leiserson. RIG column present the actual value (in seconds) of average run-time after 20 runs . Other columns have been normalized with respect to this one.

Problem	RIG	W1	W2	W3	W4	W5	W6	Hy
25fv47	0.51	0.94	0.94	0.96	0.96	0.94	0.96	2.67
80bau3b	0.87	0.94	0.95	0.94	0.94	0.94	0.94	2.38
bnl2	0.89	0.98	0.96	0.94	0.94	0.94	0.94	2.84
cycle	1.10	0.96	0.95	0.95	0.95	0.95	0.95	3.77
czprob	0.42	1.00	0.98	0.98	0.98	0.98	0.98	3.52
d2q06c	1.23	0.96	0.96	0.95	0.95	0.95	0.95	3.00
ganges	0.44	0.93	0.93	0.95	0.95	0.93	0.93	2.70
greenbea	1.81	0.94	0.91	0.92	0.91	0.90	0.91	2.63
greenbeb	1.78	0.94	0.94	0.93	0.92	0.92	0.92	2.69
scfxm3	0.41	0.95	0.95	0.98	0.95	0.95	0.93	3.56
sctap2	0.36	0.97	0.97	0.94	0.94	0.92	0.94	2.83
sctap3	0.48	0.96	0.94	0.94	0.94	0.92	0.92	2.90
ship12l	0.56	0.95	0.95	0.96	0.95	0.96	0.93	3.05
ship12s	0.34	1.00	0.97	0.97	0.97	0.97	0.97	3.50
sierra	0.37	0.95	0.95	0.92	0.92	0.89	0.92	2.68
stocfor2	0.72	0.96	0.94	0.94	0.94	0.92	0.94	3.18
woodw	0.98	0.95	0.92	0.89	0.88	0.88	0.88	2.45
cre-a	1.28	0.99	1.00	0.99	0.98	0.99	0.98	3.26
cre-c	1.14	1.01	0.98	0.96	0.94	0.95	0.96	3.11
cre-d	6.70	0.98	0.99	0.99	0.98	1.00	1.00	4.76
osa-07	4.13	0.99	0.99	1.00	0.97	0.98	0.96	2.80
CO9	4.45	1.00	1.02	1.04	1.05	1.06	1.05	6.31
CQ9	4.07	1.02	1.04	1.06	1.07	1.07	1.07	5.77
GE	1.92	1.00	1.00	1.01	0.99	1.01	1.00	5.02
NL	2.67	1.04	1.09	1.14	1.16	1.21	1.15	3.75
mod2	9.64	1.01	1.02	1.02	1.04	1.04	1.04	7.11
world	9.50	1.02	1.05	1.05	1.05	1.05	1.03	7.22
Avg	1.000	0.976	0.974	0.975	0.935	0.897	0.932	3.684

Table A.20. Comparison of separators with weighted and unweighted models UW and W stand for the unweighted and weighted models for finding wide separators. One-Max heuristic has been used for finding the narrow separator. Min and Avg are minimum and average separator sizes after 20 runs. Time is the average time of 20 runs in seconds. Columns of UW present the actual values, and columsn of W have been normalized with respect to W.

Problem	k	Min		Avg		Time	
		UW	W	UW	W	UW	W
25fv47	2	80	0.95	87.55	0.97	0.29	1.07
	4	112	1.00	127.83	1.00	0.39	1.03
	6	141	1.04	155.50	1.00	0.45	1.00
	8	181	0.91	188.80	1.00	0.51	0.94
80bau3b	2	74	0.95	85.00	0.96	0.41	1.05
	4	199	0.94	218.35	0.95	0.63	0.98
	6	291	0.90	326.40	0.88	0.79	0.95
	8	337	0.94	357.80	0.95	0.87	0.94
bnl2	2	104	1.05	116.45	1.13	0.48	0.98
	4	273	0.95	289.62	0.98	0.68	0.97
	6	359	0.91	387.86	0.95	0.82	0.94
	8	408	0.96	456.65	0.93	0.89	0.94
cycle	2	37	1.05	46.10	1.05	0.70	0.99
	4	49	1.31	65.15	1.27	0.89	0.98
	6	76	0.91	86.35	1.08	1.02	0.98
	8	96	1.04	109.40	1.16	1.10	0.95
czprob	2	27	1.00	28.80	1.00	0.23	0.96
	4	29	0.93	30.15	0.96	0.31	1.00
	6	29	0.97	31.35	0.93	0.39	1.00
	8	29	1.00	31.60	0.92	0.42	0.98
d2q06c	2	119	1.05	151.65	1.04	0.76	0.99
	4	198	1.13	233.20	1.05	0.97	0.99
	6	256	1.05	285.60	1.04	1.13	0.98
	8	296	0.99	311.40	1.03	1.23	0.95
ganges	2	25	0.96	40.70	0.98	0.23	1.00
	4	89	0.76	106.80	0.93	0.32	1.00
	6	121	0.88	147.45	0.99	0.38	1.00
	8	150	0.85	175.20	0.90	0.44	0.93
greenbea	2	82	1.02	100.85	1.10	1.05	0.99
	4	135	0.93	175.65	0.94	1.41	0.95
	6	182	1.01	206.75	1.07	1.63	0.95
	8	237	0.97	262.00	1.06	1.81	0.90

Table A.21. Comparison of separators with weighted and unweighted models (cont.d)

Problem	k	Min		Avg		Time	
		UW	W	UW	W	UW	W
greenbeb	2	80	1.09	98.70	1.19	1.06	0.97
	4	136	1.01	159.15	1.15	1.41	0.94
	6	175	1.02	208.05	1.01	1.62	0.99
	8	232	1.03	254.20	1.10	1.78	0.92
scfxm3	2	12	1.00	16.85	1.06	0.24	0.96
	4	28	0.96	36.05	1.01	0.32	0.97
	6	49	0.98	57.75	0.96	0.37	1.00
	8	77	0.94	84.00	0.93	0.41	0.95
sctap2	2	68	1.00	71.35	1.03	0.18	1.00
	4	120	1.04	129.40	1.01	0.27	0.96
	6	152	0.98	163.15	1.01	0.33	1.00
	8	177	0.99	187.75	0.98	0.36	0.92
sctap3	2	78	0.83	86.05	0.94	0.24	1.00
	4	152	0.91	161.15	0.94	0.35	1.00
	6	184	0.90	201.30	0.91	0.44	0.98
	8	217	0.87	225.45	0.92	0.48	0.92
ship12l	2	25	0.96	39.10	0.72	0.32	0.97
	4	51	0.96	65.00	0.89	0.43	1.00
	6	66	0.95	82.40	0.96	0.52	0.98
	8	79	0.99	87.05	1.01	0.56	0.96
ship12s	2	47	0.94	56.10	0.95	0.19	1.00
	4	59	0.95	72.30	1.01	0.26	1.00
	6	64	1.02	78.25	1.00	0.32	0.97
	8	74	0.95	85.30	1.01	0.34	0.97
sierra	2	49	1.02	52.85	0.97	0.18	0.94
	4	104	0.95	111.85	0.97	0.27	0.93
	6	121	0.98	138.05	0.97	0.33	0.94
	8	139	0.96	157.65	0.92	0.37	0.89
stocfor2	2	15	1.00	20.40	1.11	0.41	0.95
	4	45	0.98	67.30	0.94	0.54	0.98
	6	95	0.99	122.15	1.06	0.65	0.95
	8	104	1.15	137.35	0.97	0.72	0.92
woodw	2	134	0.42	142.00	0.59	0.57	1.00
	4	225	0.30	275.15	0.41	0.76	0.97
	6	224	0.69	287.90	0.60	0.88	0.94
	8	245	0.65	292.15	0.68	0.98	0.88

Table A.22. Comparison of separators with weighted and unweighted models (cont.d)

Problem	k	Min		Avg		Time	
		UW	W	UW	W	UW	W
cre-a	2	105	0.76	139.50	0.72	0.81	1.00
	4	167	0.67	207.55	0.64	1.05	0.98
	6	216	0.61	255.90	0.59	1.24	0.98
	8	214	0.66	249.70	0.62	1.28	0.99
cre-c	2	167	0.42	207.95	0.46	0.68	0.97
	4	209	0.49	263.20	0.46	0.92	0.97
	6	254	0.49	287.50	0.51	1.10	0.94
	8	238	0.53	287.00	0.52	1.14	0.95
cre-d	2	852	0.77	963.68	0.84	5.50	1.00
	4	1180	0.77	1325.62	0.81	6.37	0.96
	6	1362	0.74	1504.33	0.81	6.63	1.01
	8	1293	0.86	1438.55	0.86	6.70	1.00
osa-07	2	73	1.00	75.40	0.99	2.33	1.01
	4	80	1.00	80.00	1.00	3.35	1.01
	6	82	0.99	82.00	1.00	3.26	0.98
	8	80	1.00	80.90	0.99	4.13	0.98
CO9	2	1098	0.66	1281.60	0.67	3.65	1.06
	4	1933	0.57	2052.10	0.61	4.02	1.07
	6	2201	0.59	2338.05	0.60	4.32	1.06
	8	2360	0.58	2427.95	0.62	4.45	1.06
CQ9	2	1115	0.53	1271.50	0.55	3.25	1.09
	4	1684	0.45	1783.10	0.54	3.67	1.09
	6	1870	0.53	2001.10	0.56	3.95	1.08
	8	1857	0.57	2040.95	0.57	4.07	1.07
GE	2	206	0.93	242.55	1.04	1.54	0.99
	4	356	0.93	408.70	0.96	1.73	0.99
	6	470	1.01	532.10	1.02	1.88	0.97
	8	563	0.92	610.55	0.94	1.92	1.01
NL	2	1055	0.32	1103.35	0.43	1.93	1.24
	4	1448	0.38	1667.80	0.39	2.31	1.22
	6	1504	0.41	1892.10	0.41	2.58	1.18
	8	1914	0.33	2045.00	0.39	2.67	1.21
mod2	2	338	0.98	522.85	0.86	8.54	1.02
	4	572	0.98	892.20	0.77	9.05	1.04
	6	976	0.65	1187.55	0.72	9.39	1.03
	8	1314	0.78	1525.95	0.79	9.64	1.04
world	2	430	0.82	588.15	0.82	8.40	1.02
	4	643	0.96	1003.80	0.73	8.89	1.04
	6	1107	0.64	1229.85	0.69	9.29	1.02
	8	1297	0.83	1600.55	0.76	9.50	1.05

Table A.23. General Comparison of separators with weighted and unweighted models
All entries have been normalized with respect to UW.

k	Min		Avg		Time	
	UW	W	UW	W	UW	W
2	1.00	0.87	1.00	0.90	1.00	1.01
4	1.00	0.86	1.00	0.86	1.00	1.00
6	1.00	0.85	1.00	0.86	1.00	0.99
8	1.00	0.86	1.00	0.87	1.00	0.97
Avg.	1.00	0.86	1.00	0.87	1.00	0.99

Table A.24. Comparison of different models

Minimum and Average fields show the minimum and average number of coupling rows after 20 runs. Time shows the average running time in seconds. Entries in the columns of RIG are the actual values. and other values have been normalized with respect to RIG.

Problem	k	Minimum			Average			Time		
		RIG	BG	RN	RIG	BG	RN	RIG	BG	RN
25fv47	2	76	1.21	1.09	84.90	1.45	1.06	0.31	1.35	1.19
	4	112	1.36	1.04	128.07	1.49	1.09	0.40	1.62	1.68
	6	147	1.45	1.05	155.45	1.57	1.16	0.45	1.76	1.87
	8	164	1.69	1.15	189.25	1.61	1.17	0.48	1.81	1.96
80bau3b	2	70	1.91	1.06	81.53	2.99	1.12	0.43	3.30	2.77
	4	187	2.34	1.34	207.05	2.59	1.52	0.62	3.63	3.74
	6	261	2.16	1.31	285.95	2.26	1.36	0.75	3.65	4.04
	8	318	2.20	1.24	341.65	2.15	1.25	0.82	3.61	4.10
bnl2	2	109	1.02	1.11	132.03	1.02	1.09	0.47	1.81	1.28
	4	258	1.07	1.08	282.55	1.10	1.13	0.66	1.68	1.79
	6	325	1.12	1.08	368.60	1.13	1.07	0.77	1.68	1.97
	8	392	1.30	1.05	425.10	1.28	1.12	0.84	1.65	2.01
cycle	2	39	1.49	0.87	48.55	1.44	0.97	0.69	1.09	0.91
	4	64	1.72	1.47	83.00	1.83	1.64	0.87	1.06	1.43
	6	69	2.04	0.81	93.50	1.67	0.84	1.00	1.01	1.60
	8	100	2.24	1.93	127.00	2.04	1.78	1.05	1.05	1.73
czprob	2	27	10.56	6.52	28.70	11.85	7.73	0.22	4.27	3.64
	4	27	12.96	5.96	29.00	14.98	7.79	0.31	3.77	4.29
	6	28	12.00	5.43	29.20	14.00	7.90	0.39	3.36	4.31
	8	29	13.90	4.59	29.05	16.18	7.57	0.41	3.44	4.07
d2q06c	2	125	1.30	1.18	157.55	1.45	1.17	0.75	1.91	1.43
	4	223	1.39	1.13	245.80	1.43	1.15	0.96	1.70	2.32
	6	268	1.38	1.03	296.80	1.40	1.11	1.11	1.58	2.59
	8	293	1.45	1.08	321.75	1.45	1.15	1.17	1.58	2.83
ganges	2	24	1.12	1.17	39.75	0.91	0.95	0.23	1.30	1.22
	4	68	1.35	0.97	99.75	1.17	0.83	0.32	1.31	1.81
	6	107	1.25	0.79	146.00	1.09	0.73	0.38	1.34	1.95
	8	128	1.23	0.87	157.65	1.15	0.84	0.41	1.34	2.07
greenbea	2	84	1.12	1.08	111.00	1.18	0.98	1.04	1.52	1.09
	4	125	1.37	1.16	165.20	1.29	1.11	1.34	1.43	1.63
	6	183	1.16	1.05	220.40	1.37	1.12	1.55	1.34	1.87
	8	231	1.34	0.92	278.05	1.29	1.00	1.63	1.34	1.98

Table A.25. Comparison of different models (cont.d)

Problem	k	Minimum			Average			Time		
		RIG	BG	RN	RIG	BG	RN	RIG	BG	RN
greenbeb	2	87	1.05	0.94	117.10	1.03	0.91	1.03	1.50	1.12
	4	138	1.21	1.07	182.25	1.16	1.07	1.33	1.40	1.68
	6	178	1.19	0.98	210.05	1.33	1.11	1.61	1.28	1.86
	8	238	1.35	0.99	278.85	1.32	0.99	1.63	1.33	2.00
scfxm3	2	12	0.92	0.83	17.90	0.84	0.74	0.23	1.39	1.04
	4	27	1.04	0.85	36.55	0.98	0.92	0.31	1.52	1.58
	6	48	0.83	0.88	55.70	0.83	0.92	0.37	1.51	1.78
	8	72	1.21	1.06	78.30	1.30	1.08	0.39	1.64	1.90
sctap2	2	68	1.15	0.60	73.80	1.18	0.63	0.18	1.72	1.28
	4	125	1.18	0.81	130.60	1.22	0.81	0.26	1.73	1.77
	6	149	1.21	0.78	164.10	1.22	0.82	0.33	1.64	1.82
	8	176	1.16	0.82	183.55	1.23	0.86	0.33	1.79	2.12
sctap3	2	65	1.34	0.62	80.90	1.21	0.62	0.24	1.92	1.17
	4	139	1.17	0.58	151.20	1.22	0.79	0.35	1.86	1.66
	6	165	1.25	0.82	183.00	1.28	0.83	0.43	1.79	1.84
	8	189	1.31	0.86	207.65	1.27	0.87	0.44	1.98	2.00
ship12l	2	24	1.08	0.42	28.05	3.17	0.37	0.31	3.71	2.00
	4	49	0.96	0.20	58.10	2.94	0.18	0.43	3.72	2.79
	6	63	0.83	0.16	78.70	1.55	0.13	0.51	3.73	3.16
	8	78	3.23	1.64	87.75	3.25	1.67	0.54	4.02	3.37
ship12s	2	44	0.52	0.23	53.45	1.12	0.19	0.19	3.37	1.42
	4	56	0.50	0.18	73.15	0.79	0.14	0.26	3.15	2.04
	6	65	0.85	0.15	78.40	1.31	0.13	0.31	3.03	2.23
	8	70	1.97	1.27	86.10	1.99	1.15	0.33	2.97	2.39
sierra	2	50	1.00	0.84	51.45	1.07	0.98	0.17	2.41	1.41
	4	99	1.05	0.82	108.40	1.10	0.88	0.25	2.32	1.84
	6	119	1.11	0.84	133.40	1.16	0.92	0.31	2.26	1.97
	8	134	1.24	0.90	144.95	1.24	0.96	0.33	2.30	2.03
stocfor2	2	15	0.93	0.73	22.60	0.90	0.87	0.39	1.21	0.85
	4	44	0.95	0.89	63.20	0.90	0.74	0.53	1.19	1.30
	6	94	1.01	1.11	129.90	0.87	0.90	0.62	1.16	1.47
	8	120	0.79	0.78	132.90	0.88	0.81	0.66	1.20	1.58
woodw	2	56	3.91	3.43	84.00	3.57	2.41	0.57	4.07	6.44
	4	68	9.15	6.35	113.40	5.97	4.11	0.74	4.43	9.05
	6	155	5.30	2.50	173.90	5.19	2.58	0.83	4.69	9.14
	8	160	5.06	2.79	197.75	4.73	2.48	0.86	4.73	9.00

Table A.26. Comparison of different models (cont.d)

Problem	k	Minimum			Average			Time		
		RIG	BG	RN	RIG	BG	RN	RIG	BG	RN
cre-a	2	80	2.12	1.14	100.40	2.11	1.09	0.81	1.32	1.02
	4	112	2.36	1.11	133.85	2.50	1.07	1.03	1.22	1.34
	6	131	2.25	1.08	151.75	2.43	1.05	1.22	1.19	1.42
	8	141	2.44	1.03	154.55	2.58	1.07	1.27	1.17	1.48
cre-c	2	70	2.66	1.07	94.80	2.75	1.10	0.66	1.52	1.20
	4	102	3.44	1.07	122.25	3.73	1.06	0.89	1.39	1.43
	6	124	3.15	0.98	145.55	3.56	1.01	1.03	1.39	1.55
	8	127	3.45	1.00	149.40	3.71	1.01	1.08	1.38	1.62
cre-d	2	657	3.96	1.48	810.30	3.65	1.53	5.51	18.74	3.91
	4	913	3.92	1.32	1067.50	3.90	1.36	6.12	22.16	5.46
	6	1004	4.11	1.51	1214.55	3.76	1.39	6.69	21.93	5.96
	8	1117	3.81	1.35	1242.55	3.70	1.45	6.73	22.44	6.51
osa-07	2	73	6.26	13.98	74.85	8.13	13.85	2.35	15.10	4.38
	4	80	10.98	12.66	80.00	12.12	12.91	3.39	13.47	4.23
	6	81	9.88	12.59	81.95	12.03	12.68	3.21	15.52	5.23
	8	80	11.61	12.48	80.45	12.75	12.86	4.05	13.64	4.07
CO9	2	724	1.79	1.08	861.10	1.83	1.25	3.88	1.25	1.41
	4	1099	2.09	0.83	1258.95	1.93	1.19	4.30	1.38	2.22
	6	1292	2.10	0.86	1396.30	2.00	1.15	4.56	1.45	2.57
	8	1363	2.13	0.89	1501.20	1.99	1.09	4.72	1.43	2.62
CQ9	2	596	2.26	1.12	702.75	2.17	1.41	3.53	1.14	1.41
	4	751	2.89	1.13	971.75	2.36	1.28	4.00	1.25	2.15
	6	982	2.52	1.02	1113.00	2.37	1.25	4.27	1.35	2.44
	8	1061	2.52	0.80	1167.90	2.38	1.25	4.36	1.36	2.51
GE	2	192	1.49	1.08	251.45	1.32	0.94	1.52	1.31	1.27
	4	331	1.44	1.00	393.55	1.40	1.04	1.71	1.33	2.17
	6	477	1.29	1.10	542.70	1.30	1.06	1.83	1.34	2.66
	8	517	1.46	1.15	576.90	1.50	1.12	1.93	1.34	2.77
NL	2	336	2.94	1.07	472.60	2.61	0.97	2.40	0.92	0.93
	4	547	3.30	0.85	656.25	3.04	0.93	2.82	0.95	1.33
	6	623	3.30	1.02	771.60	2.88	0.91	3.04	0.99	1.53
	8	633	3.51	1.02	807.00	2.91	0.93	3.22	0.97	1.54
mod2	2	330	2.22	1.07	452.25	3.13	0.87	8.72	1.19	0.94
	4	559	1.77	1.21	691.15	2.53	1.14	9.44	1.33	1.61
	6	631	2.16	1.41	856.50	2.76	1.25	9.67	1.41	2.05
	8	1029	1.82	1.27	1200.75	1.90	1.22	10.07	1.42	2.15
world	2	354	1.89	1.05	485.15	2.44	0.84	8.61	1.26	0.98
	4	615	1.87	1.15	736.80	2.48	1.08	9.24	1.41	1.69
	6	712	2.79	1.30	846.75	2.84	1.29	9.47	1.51	2.16
	8	1074	1.79	1.23	1214.40	2.02	1.22	10.02	1.48	2.04

Table A.27. General Comparison of Different Models
All values have been normalized with respect to RIG. Results of osa-07 has been excluded for this table.

k	Min			Average			Time		
	RIG	BG	RN	RIG	BG	RN	RIG	BG	RN
2	1.00	2.03	1.26	1.00	2.24	1.26	1.00	2.52	1.68
4	1.00	2.43	1.36	1.00	2.52	1.38	1.00	2.65	2.36
6	1.00	2.28	1.20	1.00	2.41	1.30	1.00	2.62	2.59
8	1.00	2.50	1.29	1.00	2.56	1.42	1.00	2.68	2.68
Avg	1.00	2.31	1.28	1.00	2.43	1.34	1.00	2.62	2.33
Avg	1.00	2.30	1.64	1.00	2.37	1.70	1.00	3.01	2.45

B. Pictures of Matrices

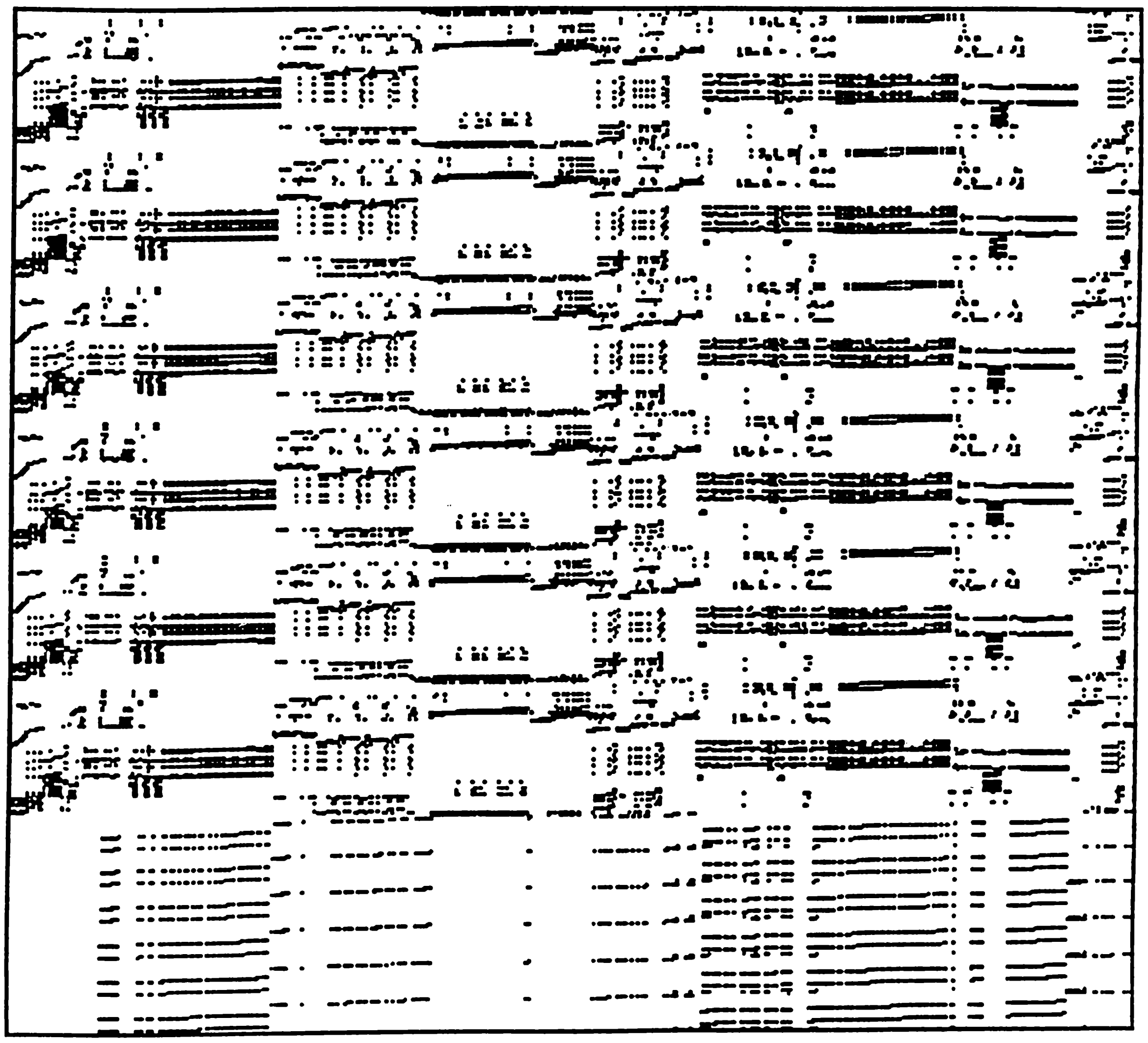


Figure B.1. Matrix GE Original Structure

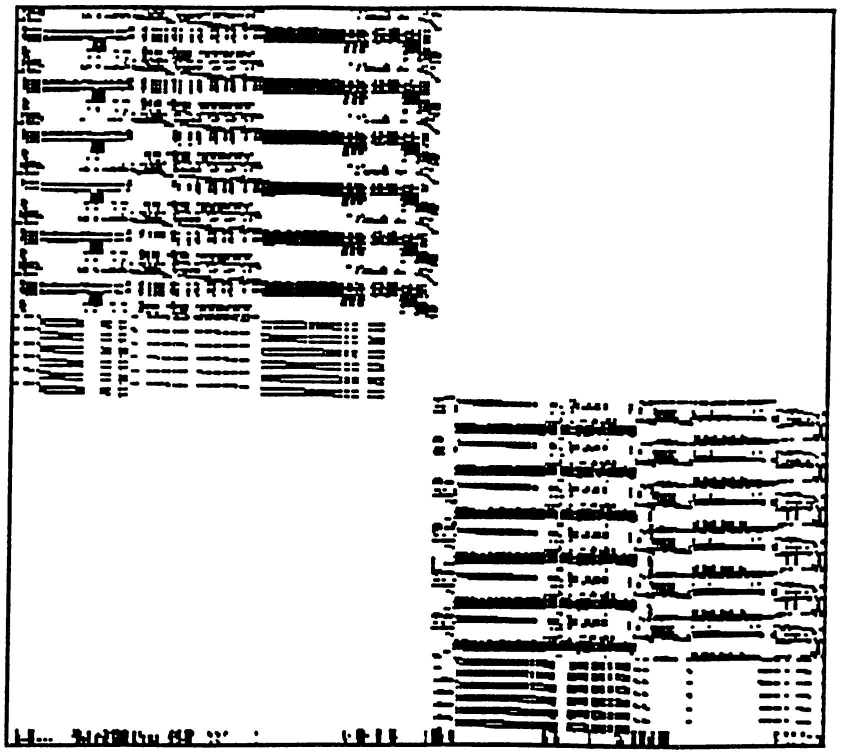


Figure B.2. Matrix GE after 2 Block Decomposition

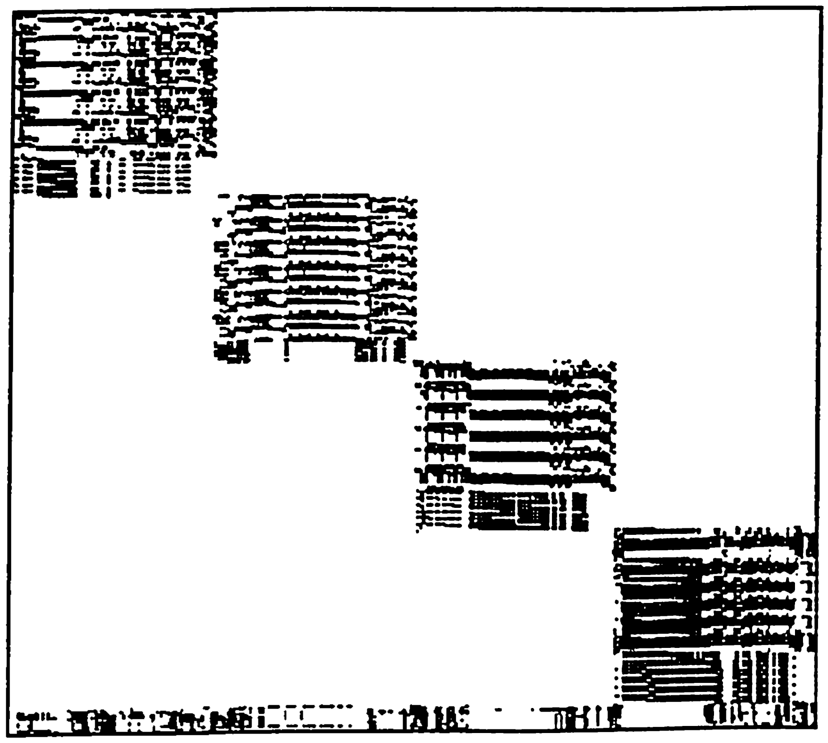


Figure B.3. Matrix GE after 4 Block Decomposition

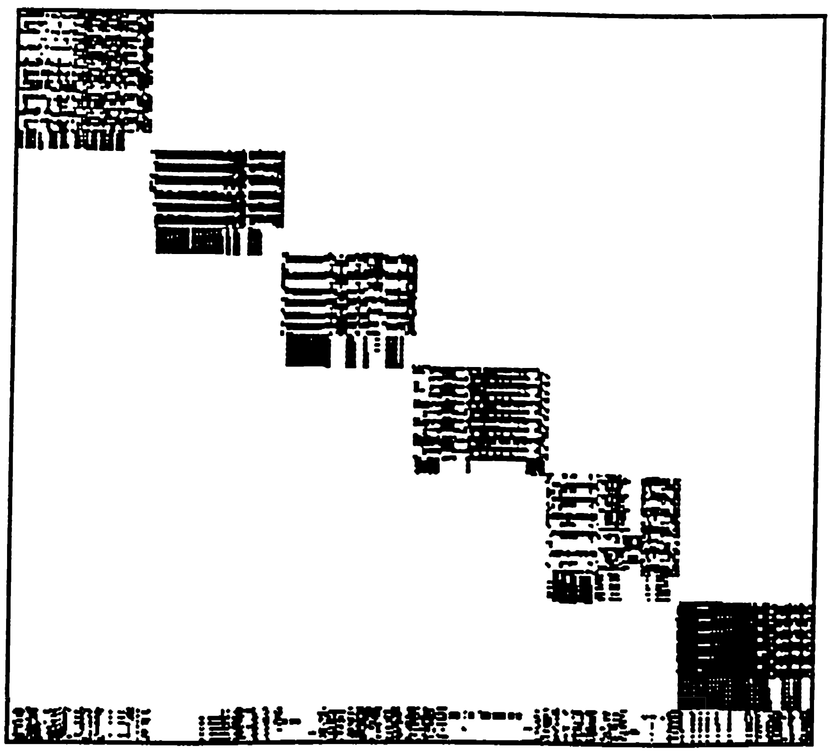


Figure B.4. Matrix GE after 6 Block Decomposition

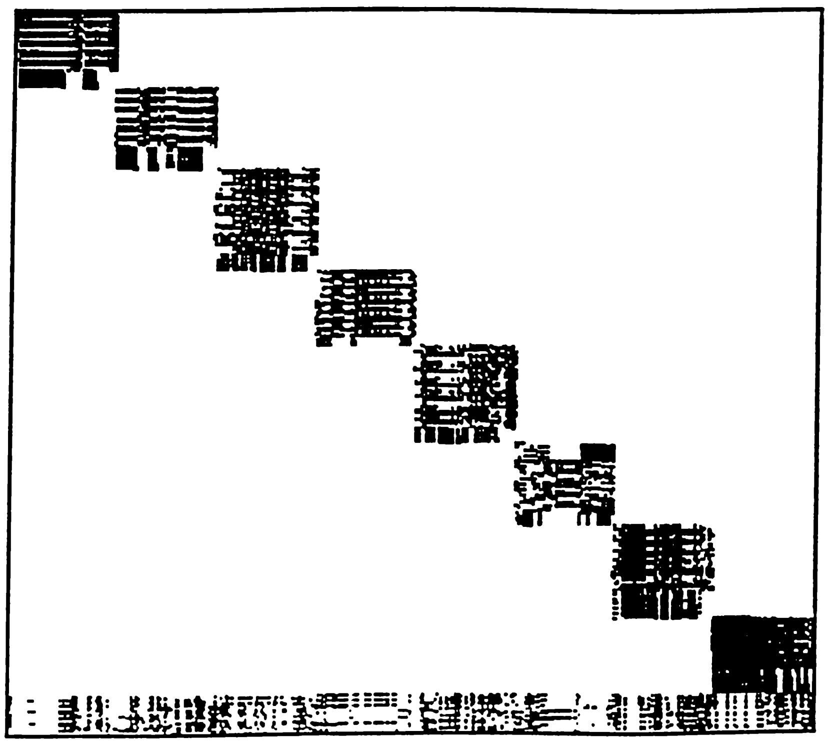


Figure B.5. Matrix GE after 8 Block Decomposition

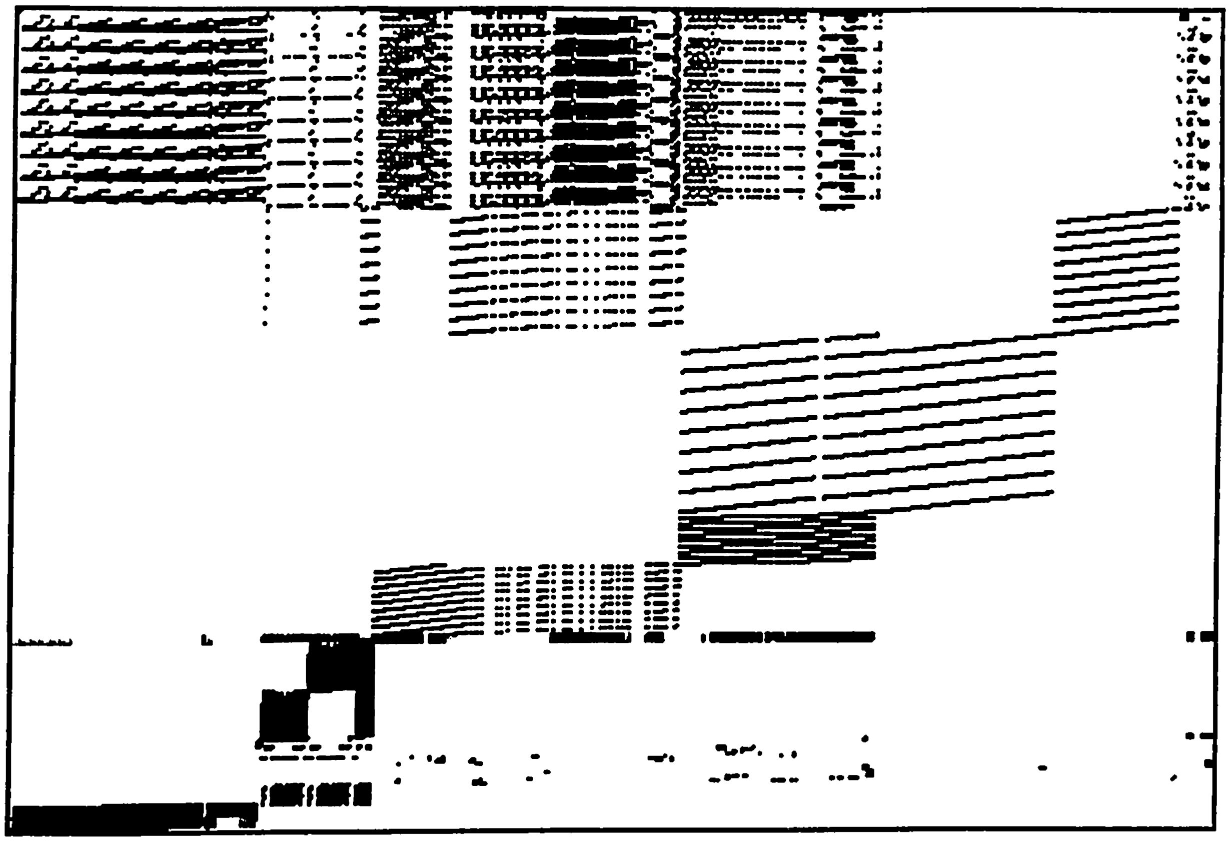


Figure B.6. Matrix CQ9 Original Structure

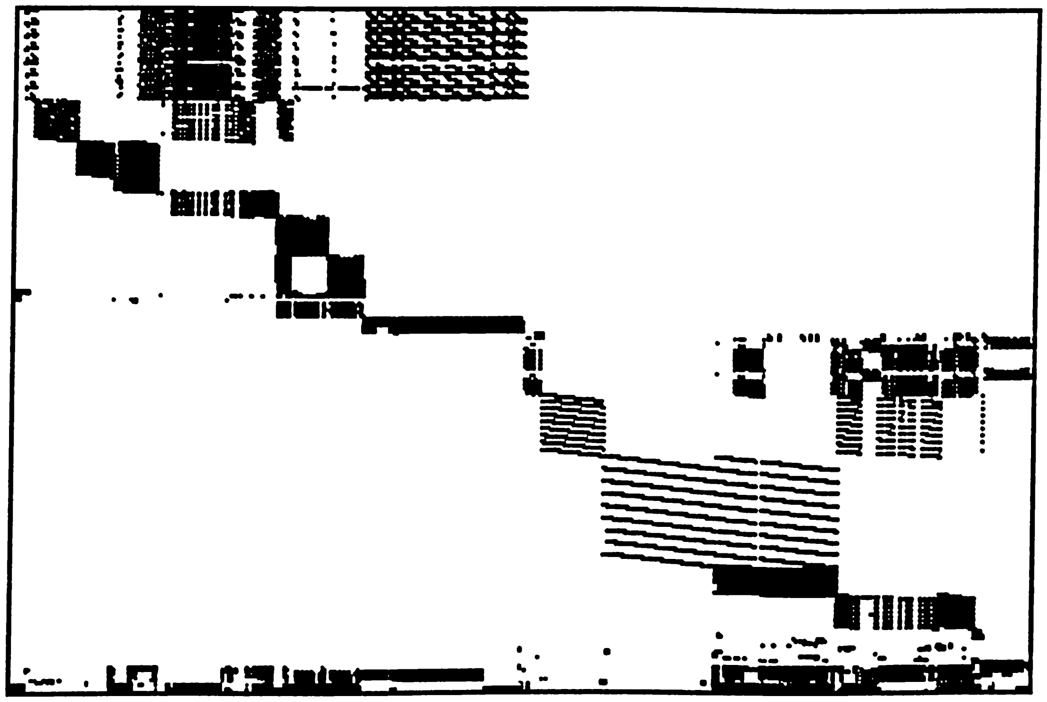


Figure B.7. Matrix CQ9 after 2 Block Decomposition

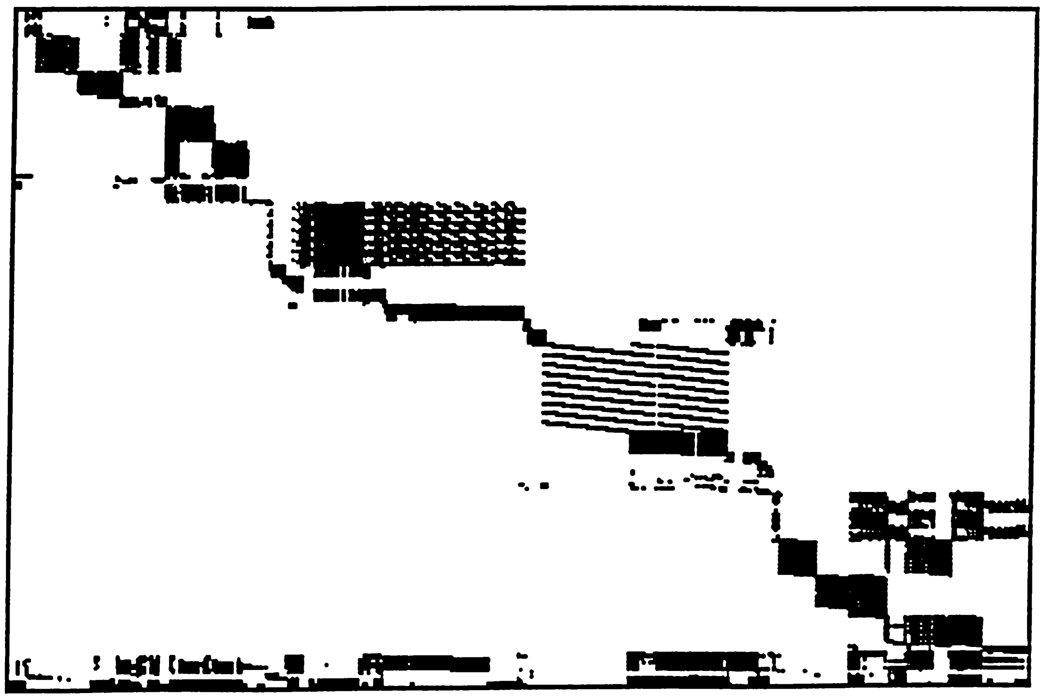


Figure B.8. Matrix CQ9 after 4 Block Decomposition

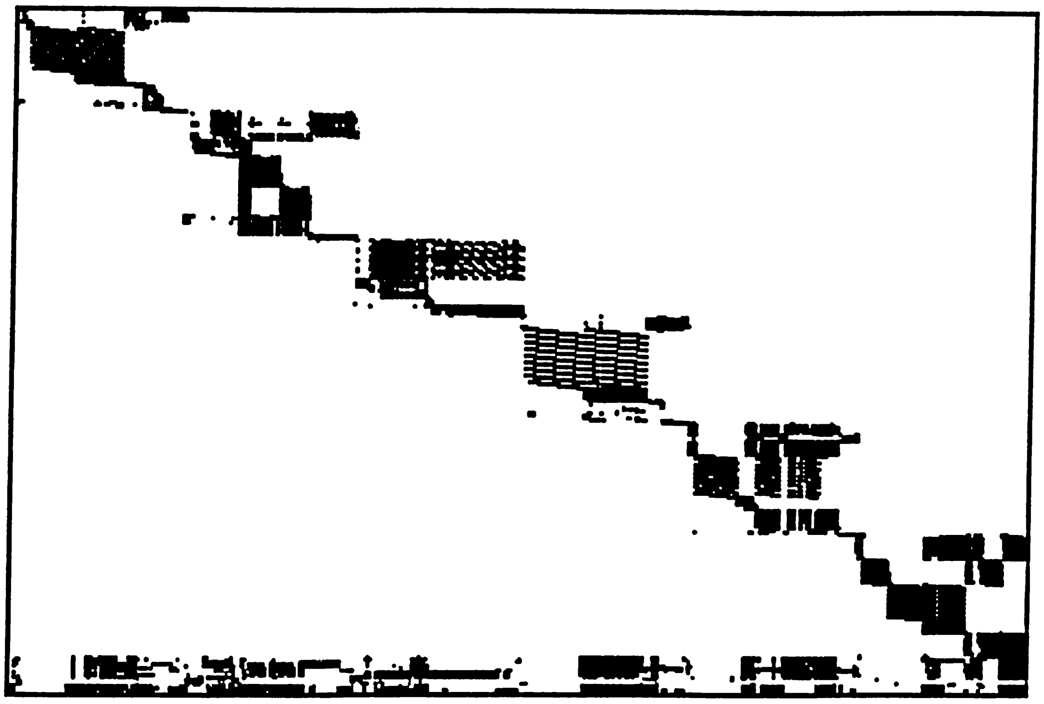


Figure B.9. Matrix CQ9 after 6 Block Decomposition

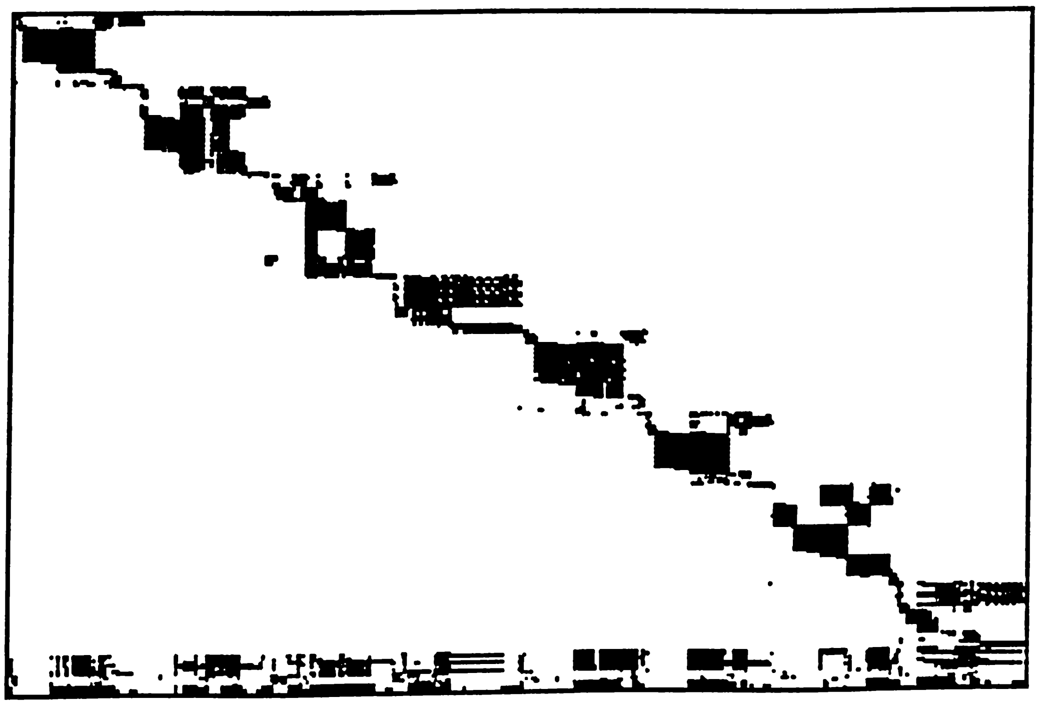


Figure B.10. Matrix CQ9 after 8 Block Decomposition