Technical Report

OSUBMI-TR-2009-n02/ BU-CE-0904

Hypergraph Partitioning-Based Fill-Reducing Ordering

Ümit V. Çatalyürek, Cevdet Aykanat and Enver Kayaaslan

April 2009

The Ohio State University
Department of Biomedical Informatics
3190 Graves Hall, 333 W. 10th Avenue
Columbus, OH 43210
http://bmi.osu.edu

Bilkent University
Computer Engineering Department
Faculty of Engineering
06800 Ankara, Turkey
http://cs.bilkent.edu.tr

HYPERGRAPH PARTITIONING-BASED FILL-REDUCING ORDERING

ÜMIT V. ÇATALYÜREK*, CEVDET AYKANAT†, AND ENVER KAYAASLAN‡

Abstract. A typical first step of a direct solver for linear system $Mx = b$ is reordering of symmetric matrix M to improve execution time and space requirements of the solution process. In this work, we propose a novel nested-dissection-based ordering approach that utilizes hypergraph partitioning. Our approach is based on formulation of graph partitioning by vertex separator (GPVS) problem as a hypergraph partitioning problem. This new formulation is immune to deficiency of GPVS in a multilevel framework hence enables better orderings. In matrix terms, our method relies on the existence of a *structural* factorization of the input M matrix in the form of $M = AA^T$ (or $M = AD^2A^T$). We show that the partitioning of the row-net hypergraph representation of rectangular matrix A induces a GPVS of the standard graph representation of matrix M. In the absence of such factorization, we also propose simple, yet effective structural factorization techniques that are based on finding an edge clique cover of the standard graph representation of matrix M, and hence applicable to any arbitrary symmetric matrix M. Our experimental evaluation has shown that the proposed method achieves better ordering in comparison to state-of-the-art graph-based ordering tools even for symmetric matrices where structural $M = AA^T$ factorization is not provided as an input. For matrices coming from linear programming problems, our method enables even faster and better orderings.

Key words. Fill-reducing ordering; hypergraph partitioning; combinatorial scientific computing.

AMS subject classifications. 05C65, 05C85, 68R10, 68W05

1. Introduction.

In most scientific computing applications, the core of the computation is solving a symmetric system of linear equations in the form $Mx=b$. Direct methods, such as LU and Cholesky factorizations, are commonly preferred for solving such systems for their numerical robustness. A typical first step of a direct method is a heuristic reordering of the rows and columns of M to reduce *fill* in the triangular factor matrices. The fill is the set of zero entries in M that become nonzero in the triangular factor matrices. Another goal in reordering is to reduce the number of floating-point operations required to perform the triangular factorization, also known as *operation count*. It is equal to the sum of the squares of the nonzeros of each eliminated row/column, hence it is directly related with the number of fills.

For a symmetric matrix, the evolution of the nonzero structure during the factorization can easily be described in terms of its graph representation [42]. In graph terms, the elimination of a vertex (which corresponds to a row/column of the matrix) creates edges for every pair of its adjacent vertices. In other words, elimination of a vertex makes its adjacent vertices clique of size its degree minus one. In this process, the added edges directly correspond to the *fill* in the matrix. Obviously, the amount of fill and operation count depends on the row/column elimination order. The aim of ordering is to reduce these quantities which leads to both faster and less memory intensive solution of the linear system. Unfortunately this problem is known to be NP-hard [46], hence we consider heuristic ordering methods.

Heuristic methods for fill reducing ordering can be divided into mainly two categories: *bottom-up* (also called *local*) and *top-down* (also called *global*) approaches [41]. In the bottom-up category, one of the most popular ordering methods is *Minimum Degree* (MD)

*Department of Biomedical Informatics, The Ohio State University, Columbus, 43210, OH (catalyurek.1@osu.edu). Supported in parts by the National Science Foundation Grants CNS-0643969, CNS-0403342 and the Department of Energy's Office of Science through the CSCAPES SciDAC Institute DE-FC02-06ER25775.

†Computer Engineering Department, Bilkent University, Ankara, Turkey (aykanat@cs.bilkent.edu.tr). Partially supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) under project EEEAG-106E069.

‡Computer Engineering Department, Bilkent University, Ankara, Turkey (enver@cs.bilkent.edu.tr).

heuristic [44] in which at every elimination step vertex with minimum degree, hence the name, is chosen for elimination. Success of the MD heuristic is followed by many variants of it, such as Quotient Minimum Degree (QMD) [25], Multiple Minimum Degree (MMD) [40], Approximate Minimum Degree (AMD) [2], and Approximate Minimum Fill (AMF) [43]. Among the top-down approaches, one the most famous and influential one is surely *nested dissection* (ND) [24]. The main idea of ND is as follows: Consider a partitioning of vertices into three sets: $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_S$, such that the removal of $\mathcal{V}_S$, called *separator*, decouples $\mathcal{V}_1$ and $\mathcal{V}_2$. If we order the vertices of $\mathcal{V}_S$ after the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$, certainly no fill can occur between the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$. Furthermore, the elimination process in $\mathcal{V}_1$ and $\mathcal{V}_2$ are independent tasks and their elimination only incurs fill to themselves and $\mathcal{V}_S$. Hence, the ordering of the vertices of $\mathcal{V}_1$ and $\mathcal{V}_2$ can be computed by applying the algorithm recursively. In ND, since the quality of the ordering depends on the size of $\mathcal{V}_S$, finding a small separator is desirable.

Although the ND scheme has some nice theoretical results [24], it has not been widely used until the development of recent multilevel graph partitioning tools. State-of-the-art ordering tools [17, 29, 32, 37] are mostly a hybrid of top-down and bottom-up approaches and built using incomplete ND approach that utilizes a multilevel graph partitioning framework [10, 28, 31, 35] for recursively identifying separators until a part becomes sufficiently small. After this point, a variant of MD, like *Constraint Minimum Degree* (CMD) [41] is used for the ordering of parts.

Some of these tools are built around graph partitioning by edge separator (GPES) frameworks [10, 35], whereas the others directly employs graph partitioning by vertex separator (GPVS) in a multilevel framework [32]. Any edge separator found by a GPES tool can be transformed into a *wide* vertex separator by including all the vertices incident to separator edges into the vertex separator. Here, a separator is said to be *wide* if a subset of it forms a separator and *narrow* otherwise. The GPES-based tools utilize algorithms like vertex cover to obtain a narrow separator from this initial wide separator. It has been shown that the GPVS-based tools outperforms the GPES-based tools [32], since the GPES-based tools do not directly aims to minimize vertex separator size. However, as we will demonstrate in §2.5, even GPVS-based approaches have a deficiency in multilevel frameworks.

In this work, we propose a new incomplete ND-based fill reducing ordering. Our approach is based on a novel formulation of the GPVS problem as a *hypergraph partitioning* (HP) problem which is immune to GPVS's deficiency in multilevel partitioning frameworks. Our formulation relies on finding an edge clique cover of the standard graph representation of matrix M. The edge clique cover is used to construct a hypergraph, which is referred to here as the clique-node hypergraph. In this hypergraph, the nodes correspond to the cliques of the edge clique cover and the hyperedges correspond to the vertices of the standard graph representation of matrix M. We show that the partitioning of the clique-node hypergraph can be decoded as a GPVS of the standard graph representation of matrix M. In matrix terms, our formulation corresponds to finding a *structural* factorization of matrix M in the form of $M = AA^T$ (or $M = AD^2A^T$). Such factorizations, luckily, exist in applications like the solution of linear programming (LP) problems using an interior point method. Furthermore, we develop matrix sparsening techniques that allow faster orderings of matrices coming from LP problems.

To the best of our knowledge, our work, including our preliminary work that had been presented in [12, 16], is the first work that utilizes hypergraph partitioning for fill reducing ordering. This paper presents a much more detailed and formal presentation of our proposed HP-based GPVS formulation in §3, and its application for fill reducing ordering symmetric matrices in §4. A recent and complimentary work [27] follows a different path and tack-

les unsymmetric ordering by leveraging our hypergraph models for permuting matrices into singly bordered block-diagonal form [8]. The HP-based fill reducing ordering method we introduce in §4 is targeted for ordering symmetric matrices and uses our proposed HP-based GPVS formulation. For general symmetric matrices, the theoretical foundations of HP-based formulation of GPVS presented in this paper lead to development of two new hypergraph construction algorithms which we present in §3.2. For matrices arising from LP problems, we present two structural factor sparsening methods in §4.2, one of which is a new formulation of the problem as a minimum set cover problem. A detailed experimental evaluation of the proposed methods is presented in §5 shows that our method achieves better orderings in comparison to the state-of-the-art ordering tools. Finally, we conclude in §6.

2. Preliminaries.

2.1. Graph partitioning by vertex separator (GPVS). An undirected graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ is defined as a set $\mathcal{V}$ of vertices and set $\mathcal{E}$ of edges. Every edge $e_{ij}\in\mathcal{E}$ connects a pair of distinct vertices v_i and v_j. We use the notation $Adj_{\mathcal{G}}(v_i)$ to denote the set of vertices that are adjacent to vertex v_i in graph $\mathcal{G}$. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}'\subseteq\mathcal{V}$, i.e., $Adj_{\mathcal{G}}(\mathcal{V}')=\{v_j\in\mathcal{V}-\mathcal{V}' : v_j\in Adj_{\mathcal{G}}(v_i) \text{ for some } v_i\in\mathcal{V}'\}$. The degree d_i of a vertex v_i is equal to the number of edges incident to v_i, i.e., $d_i=|Adj_{\mathcal{G}}(v_i)|$. A vertex subset $\mathcal{V}_S$ is a K-way *vertex separator* if the subgraph induced by the vertices in $\mathcal{V}-\mathcal{V}_S$ has at least K connected components. $\Pi_{VS}=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K;\mathcal{V}_S\}$ is a K-way vertex partition of $\mathcal{G}$ by vertex separator $\mathcal{V}_S\subseteq\mathcal{V}$ if the following conditions hold: $\mathcal{V}_k\subseteq\mathcal{V}$ and $\mathcal{V}_k\neq\emptyset$ for $1\leq k\leq K$; $\mathcal{V}_k\cap\mathcal{V}_\ell=\emptyset$ for $1\leq k\leq\ell\leq K$ and $\mathcal{V}_k\cap\mathcal{V}_S=\emptyset$ for $1\leq k\leq K$; $\bigcup_{k=1}^{K}\mathcal{V}_k\cup\mathcal{V}_S=\mathcal{V}$; removal of $\mathcal{V}_S$ gives K disconnected parts $\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K$ (i.e., $Adj_{\mathcal{G}}(\mathcal{V}_k)\subseteq\mathcal{V}_S$ for $1\leq k\leq K$).

In the GPVS problem, the partitioning constraint is to maintain a balance criteria on the weights of the K parts of the K-way vertex partition $\Pi_{VS}=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K;\mathcal{V}_S\}$. The weight W_k of a part $\mathcal{V}_k$ is usually defined by the number of the vertices in $\mathcal{V}_k$, i.e., $W_k=|\mathcal{V}_k|$, for $1\leq k\leq K$. The partitioning objective is to minimize the separator size, which is usually defined as the number of vertices in the separator, i.e.,

$$Separatorsize(\Pi_{VS})=|\mathcal{V}_S|. \tag{2.1}$$

2.2. Hypergraph partitioning (HP). A hypergraph $\mathcal{H}=(\mathcal{U},\mathcal{N})$ is defined as a set $\mathcal{U}$ of nodes (vertices) and a set $\mathcal{N}$ of nets (hyperedges). We refer to the vertices of $\mathcal{H}$ as nodes, to avoid the confusion between graphs and hypergraphs. Every net $n_i\in\mathcal{N}$ connects a subset of nodes of $\mathcal{U}$, which are called as the pins of n_i and denoted as $Pins(n_i)$. The set of nets that connect node u_h is denoted as $Nets(u_h)$. Two distinct nets n_i and n_j are said to be adjacent, if they connect at least one common node. We use the notation $Adj_{\mathcal{H}}(n_i)$ to denote the set of nets that are adjacent to n_i in $\mathcal{H}$, i.e., $Adj_{\mathcal{H}}(n_i)=\{n_j\in\mathcal{N}-\{n_i\} : Pins(n_i)\cap Pins(n_j)\neq\emptyset\}$. We extend this operator to include the adjacency set of a net subset $\mathcal{N}'\subseteq\mathcal{N}$, i.e., $Adj_{\mathcal{H}}(\mathcal{N}')=\{n_i\in\mathcal{N}-\mathcal{N}' : n_i\in Adj_{\mathcal{H}}(n_j) \text{ for some } n_j\in\mathcal{N}'\}$. The degree d_h of a node u_h is equal to the number of nets that connect u_h, i.e., $d_h=|Nets(u_h)|$. The size s_i of a net n_i is equal to the number of its pins, i.e., $s_i=|Pins(n_i)|$.

$\Pi_{HP}=\{\mathcal{U}_1,\mathcal{U}_2,\ldots,\mathcal{U}_K\}$ is a K-way node partition of $\mathcal{H}$ if the following conditions hold: $\mathcal{U}_k\subseteq\mathcal{U}$ and $\mathcal{U}_k\neq\emptyset$ for $1\leq k\leq K$; $\mathcal{U}_k\cap\mathcal{U}_\ell=\emptyset$ for $1\leq k<\ell\leq K$; $\bigcup_{k=1}^{K}\mathcal{U}_k=\mathcal{U}$. In a partition Π_{HP} of $\mathcal{H}$, a net that connects at least one node in a part is said to *connect* that part. A net n_i is said to be an internal net of a node-part $\mathcal{U}_k$, if it connects only part $\mathcal{U}_k$, i.e., $Pins(n_i)\subseteq\mathcal{U}_k$. We use $\mathcal{N}_k$ to denote the set of internal nets of node-part $\mathcal{U}_k$, for $1\leq k\leq K$. A net n_i is said to be cut (external), if it connects more than one node part. We use $\mathcal{N}_S$ to denote the set of external nets, to show that it actually forms a net separator, that is, removal of $\mathcal{N}_S$ gives at least K disconnected parts.

In the HP problem, the partitioning constraint is to maintain a balance criteria on the weights of the parts of the K-way partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$. The weight W_k of a node-part $\mathcal{U}_k$ is usually defined by the cumulative effect of the nodes in $\mathcal{U}_k$, for $1 \leq k \leq K$. However, in this work, we define W_k as the number of internal nets of node-part $\mathcal{U}_k$, i.e., $W_k = |\mathcal{N}_k|$. The partitioning objective is to minimize the cutsize defined over the external nets. There are various cutsize definitions. The relevant one used in this work is the cut-net metric, where cutsize is equal to the number of external nets, i.e.,

$$Cutsize(\Pi_{HP}) = |\mathcal{N}_S| \tag{2.2}$$

2.3. Net-intersection graph (NIG) representation of a hypergraph. The NIG representation [18], also known as *intersection graph* [1, 9], was proposed and used in the literature as a fast approximation approach for solving the HP problem [33]. In the NIG representation $\mathrm{NIG}(\mathcal{H}) = (\mathcal{V}, \mathcal{E})$ of a given hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, each vertex v_i of $\mathrm{NIG}(\mathcal{H})$ corresponds to net n_i of $\mathcal{H}$. There exist an edge between vertices v_i and v_j of $\mathrm{NIG}(\mathcal{H})$ if and only if the respective nets n_i and n_j are adjacent in $\mathcal{H}$, i.e., $e_{i,j} \in \mathcal{E}$ if and only if $n_j \in Adj_{\mathcal{H}}(n_i)$ which also implies that $n_i \in Adj_{\mathcal{H}}(n_j)$. This NIG definition implies that every node u_h of $\mathcal{H}$ induces a clique $\mathcal{C}_h$ in $\mathrm{NIG}(\mathcal{H})$ where $\mathcal{C}_h = Nets(u_h)$.

2.4. Graph and hypergraph models for representing sparse matrices. Several graph and hypergraph models are proposed and used in the literature, for representing sparse matrices for a variety of applications in parallel and scientific computing [30].

In the standard graph model, a square and symmetric matrix $M = \{m_{ij}\}$ is represented as an undirected graph $\mathrm{G}(M) = (\mathcal{V}, \mathcal{E})$. Vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ respectively correspond to the rows/columns and off-diagonal nonzeros of matrix M. There exists one vertex v_i for each row/column r_i/c_i. There exists an edge e_{ij} for each symmetric nonzero pair m_{ij} and m_{ji}, i.e., $e_{ij} \in \mathcal{E}$ if $m_{ij} \neq 0$.

Three hypergraph models are proposed and used in the literature; namely row-net, column-net, and row-column-net (a.k.a. fine-grain) hypergraph models [11, 13, 15, 45]. We will only discuss the row-net hypergraph model which is relevant to our work. In the row-net hypergraph model, a rectangular matrix $A = \{a_{ij}\}$ is represented as a hypergraph $\mathrm{H}_{\mathrm{RN}}(A) = (\mathcal{U}, \mathcal{N})$. Node set $\mathcal{U}$ and net set $\mathcal{N}$ respectively correspond to the columns and rows of matrix A. There exist one node u_h for each column c_h and one net n_i for each row r_i. Net n_i connects the nodes corresponding to the columns that have a nonzero entry in row i, i.e., $c_h \in Pins(n_i)$ if $a_{ih} \neq 0$.

2.5. Deficiency of GPVS in multilevel framework. A multilevel graph/hypergraph partitioning framework basically contains three phases; coarsening, initial partitioning and uncoarsening. During the coarsening phase, vertices/nodes are visited in some (possibly random) order and usually two (or more) of them are coalesced to construct the vertices/nodes of the next-level coarsened graph/hypergraph. After multiple coarsening levels, an initial partition is found on the coarsest graph/hypergraph, and this partition is projected back to the original graph/hypergraph in the uncoarsening phase with further refinements at each level of uncoarsening. Both GPES and HP problems are well suited for the multilevel framework, because the following nice property holds for the edge and net separators in multilevel GPES and HP: Any edge/net separator at a given level of uncoarsening forms a valid *narrow* edge/net separator of all the finer graphs/hypergraphs, including the original graph/hypergraph. Here, an edge/net separator is said to be *narrow*, if no subset of edges/nets of the separator forms a separator.

However, this property does not hold for the GPVS problem. Consider the two examples displayed in Figure 2.1 as partial illustration of two different GPVS partitioning results at

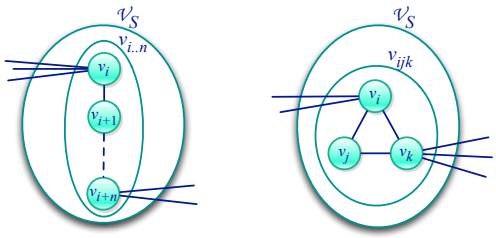


FIG. 2.1. *Partial illustration of two sample GPVS results to demonstrate the deficiency of the graph model in multilevel framework.*

some level m of a multilevel GPVS tool. In the first one, $n+1$ vertices $\{v_i, v_{i+1}, \ldots, v_{i+n}\}$ are coalesced to construct vertex $v_{i..n}$ as a result of one or more levels of coarsening. This, $\mathcal{V}_S = \{v_{i..n}\}$, is a valid and narrow vertex separator for level m. The GPVS tool computes the cost of this separator as $n+1$ at this level. However, obviously this separator is a wide separator of the original graph. In other words, there is a subset of those vertices which is a valid narrow separator of the original graph. In fact, any single vertex in $\{v_i, v_{i+1}, \ldots, v_{i+n}\}$ is a valid separator of size 1 of the original graph. Similarly, for the second example, the GPVS tool computes the size of the separator as 3, however, there is a subset of constituent vertices of $\mathcal{V}_S = \{v_{ijk}\} = \{v_i, v_j, v_k\}$ which is a valid narrow separator of size 1 in the original graph That is, either $\mathcal{V}_S = \{v_i\}$ or $\mathcal{V}_S = \{v_k\}$ is a valid narrow separator.

3. HP-based GPVS Formulation. We are considering to solve the GPVS problem for a given undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

3.1. Theoretical foundations. The following theorem lays down the basis for our HP-based GPVS formulation.

THEOREM 1. *Consider a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ and its NIG representation* $\mathrm{NIG}(\mathcal{H})$ $= (\mathcal{V}, \mathcal{E})$. *A K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ can be decoded as a K-way vertex separator $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of* $\mathrm{NIG}(\mathcal{H})$, *where*

(a) partitioning objective of minimizing the cutsize of Π_{HP} according to (2.2) corresponds to minimizing the separator size of Π_{VS} according to (2.1).

(b) partitioning constraint of balancing on the internal net counts of node parts of Π_{HP} infers balance among the vertex counts of parts of Π_{VS}

Proof. As described in [8], the K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ can be decoded as a $(K+1)$-way net-partition $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$. We consider this $(K+1)$-way net-partition $\Pi_{HP} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$ of $\mathcal{H}$ as inducing a K-way GPVS $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ on $\mathrm{NIG}(\mathcal{H})$, where $\mathcal{V}_k \equiv \mathcal{N}_k$, for $1 \leq k \leq K$, and $\mathcal{V}_S \equiv \mathcal{N}_S$. Consider an internal net n_j of node-part $\mathcal{U}_k$ in Π_{HP}, i.e., $n_j \in \mathcal{N}_k$. It is clear that $Adj_{\mathcal{H}}(n_j) \subseteq \mathcal{N}_k \cup \mathcal{N}_S$, which implies $Adj_{\mathcal{H}}(\mathcal{N}_k) \subseteq \mathcal{N}_S$. Since $\mathcal{V}_k \equiv \mathcal{N}_k$ and $\mathcal{V}_S \equiv \mathcal{N}_S$, $Adj_{\mathcal{H}}(\mathcal{N}_k) \subseteq \mathcal{N}_S$ in Π_{HP} implies $Adj_{\mathcal{G}}(\mathcal{V}_k) \subseteq \mathcal{V}_S$ in Π_{VS}. In other words, $Adj_{\mathcal{G}}(\mathcal{V}_k) \cap \mathcal{V}_\ell = \emptyset$, for $1 \leq \ell \leq K$ and $\ell \neq k$. Thus, $\mathcal{V}_S$ of Π_{VS} constitutes a valid separator of size $|\mathcal{V}_S| = |\mathcal{N}_S|$. So, minimizing the cutsize of Π_{HP} corresponds to minimizing the separator size of Π_{VS}. Since $|\mathcal{V}_k| = |\mathcal{N}_k|$, for $1 \leq k \leq K$, balancing on the internal net counts of node parts of Π_{HP} corresponds to balancing the vertex counts of parts of Π_{VS}. □

COROLLARY 1. *Consider an undirected graph $\mathcal{G}$. A K-way partition Π_{HP} of any hypergraph $\mathcal{H}$ for which* $\mathrm{NIG}(\mathcal{H}) \equiv \mathcal{G}$ *can be decoded as a K-way vertex separator Π_{VS} of $\mathcal{G}$.*

Although $\mathrm{NIG}(\mathcal{H})$ is well-defined for a given hypergraph $\mathcal{H}$, there is no unique reverse construction. We introduce the following definitions and theorems which show our approach for reverse construction.

DEFINITION 3.1 (Edge clique cover (ECC) [39]). *Given a set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ of cliques in $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, $\mathcal{C}$ is an ECC of $\mathcal{G}$ if for each edge $e_{ij} \in \mathcal{E}$ there exists a clique $\mathcal{C}_h \in \mathcal{C}$ that contains both v_i and v_j.*

DEFINITION 3.2 (Clique-node hypergraph). *Given a set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots\}$ of cliques in graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the clique-node hypergraph $\mathrm{CNH}(\mathcal{G}, \mathcal{C}) = \mathcal{H} = (\mathcal{U}, \mathcal{N})$ of $\mathcal{G}$ for $\mathcal{C}$ is defined as a hypergraph with $|\mathcal{C}|$ nodes and $|\mathcal{V}|$ nets, where $\mathcal{H}$ contains one node u_h for each clique $\mathcal{C}_h$ of $\mathcal{C}$ and one net n_i for each vertex v_i of $\mathcal{V}$, i.e., $\mathcal{U} \equiv \mathcal{C}$ and $\mathcal{N} \equiv \mathcal{V}$. In $\mathcal{H}$, the set of nets that connect node u_h corresponds to the set $\mathcal{C}_h$ of vertices, i.e., $Nets(u_h) \equiv \mathcal{C}_h$ for $1 \leq h \leq |\mathcal{C}|$. In other words, the net n_i connects the nodes corresponding to the cliques that contain vertex v_i of $\mathcal{G}$.*

Figure 3.1(a) displays a sample graph $\mathcal{G}$ with 11 vertices and 18 edges. Figure 3.1(b) shows the clique-node hypergraph $\mathcal{H}$ of $\mathcal{G}$ for a sample ECC $\mathcal{C}$ that contains 12 cliques. Note that $\mathcal{H}$ contains 12 nodes and 11 nets. As seen in Figure 3.1(b), the 4-clique $\mathcal{C}_5 = \{v_4, v_5, v_{10}, v_{11}\}$ in $\mathcal{C}$ induces node u_5 with $Nets(u_5) = \{n_4, n_5, n_{10}, n_{11}\}$ in $\mathcal{H}$. Figure 3.2(a) shows a 3-way partition Π_{HP} of $\mathcal{H}$, where each node part contains 3 internal nets and the cut contains 2 external nets. Figure 3.2(b) shows the 3-way GPVS Π_{VS} induced by Π_{HP}. In Π_{VS}, each part contains 3 vertices and the separator contains 2 vertices. In particular, the cut with 2 external nets n_{10} and n_{11} induces a separator with 2 vertices v_{10} and v_{11}. The node-part $\mathcal{U}_1$ with 3 internal nets n_1, n_2 and n_3 induces a vertex-part $\mathcal{V}_1$ with 3 vertices v_1, v_2 and v_3.

The following two theorems state that, for a given graph $\mathcal{G}$, the problem of constructing a hypergraph whose NIG representation is same as $\mathcal{G}$ is equivalent to the problem of finding an ECC of $\mathcal{G}$.

THEOREM 2. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, if $\mathrm{NIG}(\mathcal{H}) \equiv \mathcal{G}$ then $\mathcal{H} \equiv \mathrm{CNH}(\mathcal{G}, \mathcal{C})$ with $\mathcal{C} = \{\mathcal{C}_h \equiv Nets(u_h) : 1 \leq h \leq |\mathcal{U}|\}$ is an ECC of $\mathcal{G}$.*

Proof. Since $\mathrm{NIG}(\mathcal{H}) \equiv \mathcal{G}$, there is an edge $e_{ij} = \{v_i, v_j\}$ in $\mathcal{G}$ if and only if nets n_i and n_j are adjacent in $\mathcal{H}$ which means there exists a node u_h in $\mathcal{H}$ such that both $n_i \in Nets(u_h)$ and $n_j \in Nets(u_h)$. Since u_h induces the clique $\mathcal{C}_h \in \mathcal{C}$, $\mathcal{C}_h$ contains both vertices v_i and v_j. □

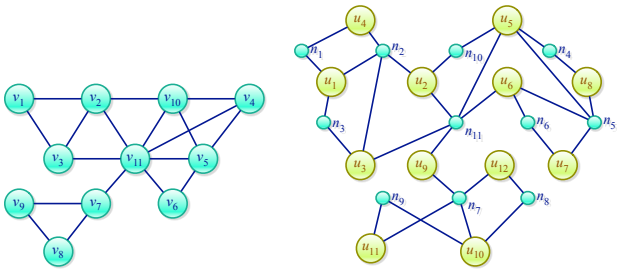
Note that $\mathcal{C} = \{\mathcal{C}_h \equiv Nets(u_h) : 1 \leq h \leq |\mathcal{U}|\}$ is the unique ECC of $\mathcal{G}$ satisfying $\mathcal{H} \equiv \mathrm{CNH}(\mathcal{G}, \mathcal{C})$.

THEOREM 3. *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, for any ECC $\mathcal{C}$ of $\mathcal{G}$, the NIG representation of clique-node hypergraph of $\mathcal{C}$ is equivalent to $\mathcal{G}$, i.e., $\mathrm{NIG}(\mathrm{CNH}(\mathcal{G}, \mathcal{C})) \equiv \mathcal{G}$.*

Proof. By construction, two nets n_i and n_j are adjacent in $\mathrm{CNH}(\mathcal{G}, \mathcal{C})$ if and only if there exists a clique $\mathcal{C}_h \in \mathcal{C}$ such that $\mathcal{C}_h$ contains both vertices v_i and v_j in $\mathcal{G}$. Since $\mathcal{C}$ is an ECC of $\mathcal{G}$, there is such a clique $\mathcal{C}_h \in \mathcal{C}$ if and only if there is an edge e_{ij} in $\mathcal{G}$. □

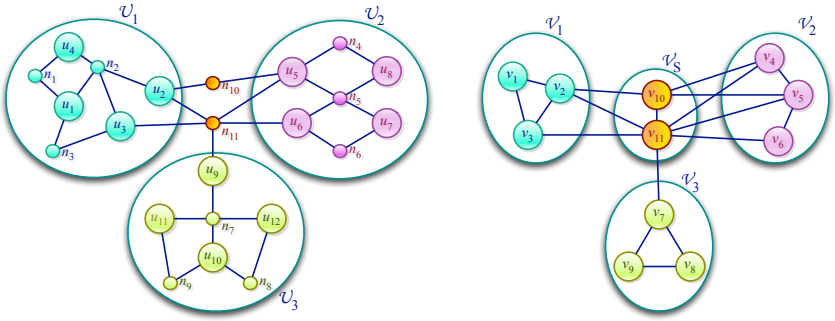
3.2. Hypergraph construction based on edge clique cover. According to the theoretical findings given in § 3.1, our HP-based GPVS approach is based on finding an ECC of the given graph and then partitioning the respective clique-node hypergraph. Here, we will briefly discuss the effects of different ECCs on the solution quality and the run-time performance of our approach.

In terms of solution quality of hypergraph partitioning, it is not easy to quantify the metrics for a "good" ECC. In a multilevel HP-tool that balances internal net weights, the choice of an ECC should not affect the quality performance of the FM-like [22] refinement heuristics commonly used in the uncoarsening phase. However, the choice of an ECC may considerably affect the quality performance of the node matchings performed in the coarsening phase.



(a) (b)

FIG. 3.1. *(a) A sample graph $\mathcal{G}$; (b) the clique-node hypergraph $\mathcal{H}$ of $\mathcal{G}$ for ECC* $\mathcal{C} = \{\mathcal{C}_1 = \{v_1, v_2, v_3\}, \mathcal{C}_2 = \{v_2, v_{10}, v_{11}\}, \mathcal{C}_3 = \{v_2, v_3, v_{11}\}, \mathcal{C}_4 = \{v_1, v_2\}, \mathcal{C}_5 = \{v_4, v_5, v_{10}, v_{11}\}, \mathcal{C}_6 = \{v_5, v_6, v_{11}\}, \mathcal{C}_7 = \{v_5, v_6\}, \mathcal{C}_8 = \{v_4, v_5\}, \mathcal{C}_9 = \{v_7, v_9\}, \mathcal{C}_{10} = \{v_7, v_8, v_9\}, \mathcal{C}_{11} = \{v_7, v_{11}\}, \mathcal{C}_{12} = \{v_7, v_8\}\}$



(a) (b)

FIG. 3.2. *(a) A 3-way partition Π_{HP} of the clique-node hypergraph $\mathcal{H}$ given in Fig. 3.1(b); (b) the 3-way GPVS Π_{VS} of G (given in Fig. 3.1(a)) induced by Π_{HP}*

For example, large cliques in the ECC may lead to better quality node matchings even in the initial coarsening levels. On the other side, large amounts of edge overlaps among the cliques of a given ECC may adversely affect the quality of the node matchings. Therefore, having large but non-overlapping cliques might be desirable for solution quality.

The choice of the ECC may affect the run-time performance of HP-tool depending on the size of the clique-node hypergraph. Since the number of nets in the clique-node hypergraph is fixed, the number of cliques and the sum of the clique sizes, which respectively correspond to the number of nodes and pins, determine the size of the hypergraph. Hence, an ECC with small number of large cliques is likely to induce a clique-node hypergraph of small size.

Although not a perfect match, the ECC problem [39], which is stated as finding an ECC with minimum number of cliques, can be considered to be relevant to our problem of finding a "good" ECC. Unfortunately, the ECC problem is also known to be NP-hard [39]. The literature contains a number of heuristics [26, 38, 39] for solving the ECC problem. However, even the fastest heuristic's [26] running time complexity is $O(|\mathcal{V}||\mathcal{E}|)$, which makes it impractical in our approach.

In this work, we investigate three different types of ECCs, namely $\mathcal{C}^2, \mathcal{C}^3$, and $\mathcal{C}^4$, to observe the effects of increasing clique size in the solution quality and run-time performance of the proposed approach. Here, $\mathcal{C}^2$ denotes the ECC of all 2-cliques (edges), i.e., $\mathcal{C}^2 = \mathcal{E}$; $\mathcal{C}^3$ denotes an ECC of 2- and 3-cliques; $\mathcal{C}^4$ denotes an ECC of 2-, 3-, and 4-cliques. In

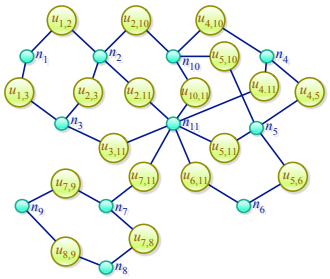


FIG. 3.3. *The 2-clique-node hypergraph $\mathcal{H}^2$ of graph $\mathcal{G}$ given in Fig. 3.1(a)*

general, $\mathcal{C}^k$ denotes an ECC of cliques in which maximum clique size is bounded above by k. Note that $\mathcal{C}^2$ is unique, whereas $\mathcal{C}^3$ and $\mathcal{C}^4$ are not necessarily unique. We will refer to the clique-node hypergraph induced by $\mathcal{C}^k$ as $\mathcal{H}^k = \mathrm{CNH}(\mathcal{G}, \mathcal{C}^k)$.

The clique-node hypergraph $\mathcal{H}^2$ deserves special attention, since it is uniquely defined for a given graph $\mathcal{G}$. In $\mathcal{H}^2$, there exists one node of degree 2 for each edge e_{ij} of $\mathcal{G}$. The net n_i corresponding to vertex v_i of $\mathcal{G}$ connects all nodes corresponding to the edges that are incident to vertex v_i, for $1 \leq i \leq |\mathcal{V}|$. So, $\mathcal{H}^2$ contains $|\mathcal{E}|$ nodes, $|\mathcal{V}|$ nets, and $2|\mathcal{E}|$ pins. The running time of HP-based GPVS using $\mathcal{H}^2$ is expected to be quite high because of the large number of nodes and pins. Figure 3.3 displays the 2-clique-node hypergraph $\mathcal{H}^2$ of the sample graph $\mathcal{G}$ given in Figure 3.1(a). As seen in the figure, each node of $\mathcal{H}^2$ is labeled as u_{ij} to show the one-to-one correspondence between nodes of $\mathcal{H}^2$ and edges of $\mathcal{G}$. That is, node u_{ij} of $\mathcal{H}^2$ corresponds to edge e_{ij} of $\mathcal{G}$, where $Nets(u_{ij}) = \{n_i, n_j\}$.

Algorithm 1 and Algorithm 2 display the algorithms developed for constructing a $\mathcal{C}^3$ and a $\mathcal{C}^4$, respectively. The goal of both algorithms is to minimize the number of pins in the clique-node hypergraphs as much as possible. Both algorithms visit the vertices in random order in order to introduce randomization to the ECC construction process. In both algorithms, each edge is processed along only one direction (i.e., from low to high numbered vertex) to avoid identifying the same clique more than once.

In Algorithm 1, for each visited vertex v_i, the 3-clique(s) that contain v_i are searched for by trying to locate 2-cliques between the vertices in $Adj_{\mathcal{G}}(v_i)$. This search is performed by scanning the adjacency list of each vertex v_j in $Adj_{\mathcal{G}}(v_i)$. For each vertex, a parent field π_1 is maintained for efficient identification of 3-cliques during this search. An identified 3-clique $\mathcal{C}_h$ is selected for inclusion in $\mathcal{C}^3$ if the number of already covered edges of $\mathcal{C}_h$ is at most 1. The rationale behind this selection criteria is as follows: Recall that a 3-clique in $\mathcal{C}^3$ adds 3 pins to $\mathcal{H}^3$, since it incurs a node of degree 3 in $\mathcal{H}^3$. If only one edge of $\mathcal{C}_h$ is already covered by other 3-clique(s) in $\mathcal{C}^3$, it is still beneficial to cover the remaining two edges of $\mathcal{C}_h$ by selecting $\mathcal{C}_h$ instead of selecting the two 2-cliques covering those uncovered edges, because the former selection incurs 3 pins whereas the latter incurs 4 pins. If, however, any two edges of $\mathcal{C}_h$ are already covered by another 3-clique in $\mathcal{C}^3$, it is clear that the remaining uncovered edge is better to be covered by a 2-clique. After scanning the adjacency list of v_j in $Adj_{\mathcal{G}}(v_i)$, if edge $\{v_i, v_j\}$ is not covered by any 3-clique then it is added to $\mathcal{C}^3$ as a 2-clique.

In Algorithm 2, for each visited vertex v_i, the 4-clique(s) that contain v_i are searched for after finding the 3-clique(s) that contain v_i as in Algorithm 1. For each 3-clique $\{v_i, v_j, v_k\}$ identified in $Adj_{\mathcal{G}}(v_i)$, the 4-clique(s) that contain the 3-clique $\{v_i, v_j, v_k\}$ are searched for by scanning $Adj_{\mathcal{G}}(v_k)$, where v_k is the last vertex added to the 3-clique. For each vertex, a second parent field π_2 is maintained together with π_1 for efficient identification of a vertex

Algorithm 1 $\mathcal{C}^3$ Construction Algorithm

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
1: **for each** vertex $v \in \mathcal{V}$ **do**
2: $\pi_1[v] \leftarrow NIL$
3: **for each** edge $e_{ij} \in \mathcal{E}$ **do**
4: cover$[e_{ij}] \leftarrow 0$
5: $\mathcal{C}^3 \leftarrow \emptyset$
6: **for each** vertex $v_i \in \mathcal{V}$ **do**
7: **for each** vertex $v_j \in Adj_{\mathcal{G}}(v_i)$ with $j > i$ **do**
8: $\pi_1[v_j] \leftarrow v_i$
9: **for each** vertex $v_j \in Adj_{\mathcal{G}}(v_i)$ with $j > i$ **do**
10: **for each** vertex $v_k \in Adj_{\mathcal{G}}(v_j)$ with $k > j$ **do**
11: **if** $\pi_1[v_k] = v_i$ **then**
12: **if** $\sum_{e \in \binom{\{v_i, v_j, v_k\}}{2}}$ cover$[e] < 2$ **then**
13: $\mathcal{C}^3 \leftarrow \mathcal{C}^3 \cup \{\{v_i, v_j, v_k\}\}$ ▷ Add the 3-clique to $\mathcal{C}^3$
14: **for each** edge $e \in \binom{\{v_i, v_j, v_k\}}{2}$ **do**
15: cover$[e] \leftarrow 1$
16: **if** cover$[e_{ij}] = 0$ **then**
17: $\mathcal{C}^3 \leftarrow \mathcal{C}^3 \cup \{\{v_i, v_j\}\}$ ▷ Add the 2-clique to $\mathcal{C}^3$
18: cover$[e_{ij}] \leftarrow 1$

v_ℓ in $Adj_{\mathcal{G}}(v_k)$ that is adjacent to both v_j and v_i. A 4-clique $\mathcal{C}_h = \{v_i, v_j, v_k, v_\ell\}$ is selected to be added to $\mathcal{C}^4$, if at most 3 out of 6 edges of $\mathcal{C}_h$ are already covered by other 3- and/or 4-clique(s) in $\mathcal{C}^4$. After scanning $Adj_{\mathcal{G}}(v_k)$, if at most one edge of the 3-clique $\{v_i, v_j, v_k\}$ is covered by other 3- or 4-cliques then it is added to $\mathcal{C}^4$ as a 3-clique. After scanning $Adj_{\mathcal{G}}(v_j)$, if edge $\{v_i, v_j\}$ is not covered by any 3- or 4-clique then it is added to $\mathcal{C}^4$ as a 2-clique.

We should note here that the ideas in Algorithms 1 and 2 can be extended to a general approach for constructing $\mathcal{C}_k$. However, this general approach requires maintaining $k-2$ parent fields for each vertex.

3.3. Matrix-theoretic view of HP-based GPVS formulation. Here, we will try to reveal the association between the graph-theoretic and matrix-theoretic views of our HP-based GPVS formulation. Given a $p \times p$ symmetric and square matrix M, let $\mathcal{G}(M) = (\mathcal{V}, \mathcal{E})$ denote the standard graph representation of matrix M.

A K-way GPVS $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of $\mathrm{G}(M)$ can be decoded as permuting matrix M into a doubly-bordered block diagonal (DB) form $M_{DB} = PAP^T$ as follows: Π_{VS} is used to define the partial row/column permutation matrix P by permuting the rows/columns corresponding to the vertices of $\mathcal{V}_k$ after those corresponding to the vertices of $\mathcal{V}_{k-1}$ for $2 \leq k \leq K$, and permuting the rows/columns corresponding to the separator vertices to the end. The partitioning objective of minimizing the separator size of Π_{VS} corresponds to minimizing the number of coupling rows/columns in M_{DB}, whereas the partitioning constraint of maintaining balance on the part weights of Π_{VS} infers balance among the row/column counts of the square diagonal submatrices in M_{DB}.

In graph-theoretic discussion given in § 3.2, we are looking for a hypergraph $\mathcal{H}$ whose NIG representation is equivalent to $\mathrm{G}(M)$. In matrix-theoretic view, this corresponds to looking for a structural factorization $M = AA^T$ of matrix M, where A is an $p \times q$ rectangular matrix. Here, structural factorization refers to the fact that $A = \{a_{ij}\}$ is a {0,1}-matrix, where AA^T determines the sparsity patterns of M. In this factorization, the rows of matrix A correspond to the vertices of $\mathrm{G}(M)$ and the set of columns of matrix A determines an

Algorithm 2 $\mathcal{C}^4$ Construction Algorithm

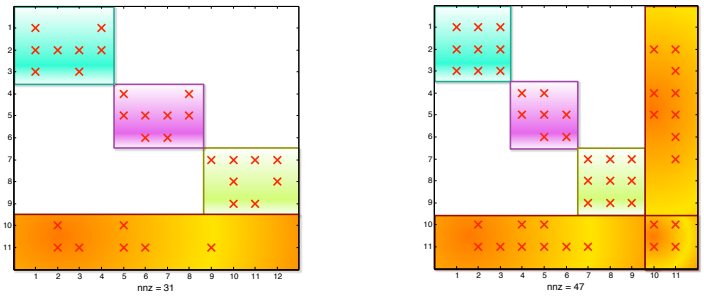
Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
for each vertex $v \in \mathcal{V}$ **do**
  $\pi_1[v] \leftarrow \pi_2[v] \leftarrow NIL$
for each edge $e_{ij} \in \mathcal{E}$ **do**
  cover$[e_{ij}] \leftarrow 0$
$\mathcal{C}^4 \leftarrow \emptyset$
for each vertex $v_i \in \mathcal{V}$ **do**
  for each vertex $v_j \in Adj_{\mathcal{G}}(v_i)$ with $j > i$ **do**
    $\pi_1[v_j] \leftarrow v_i$
  for each vertex $v_j \in Adj_{\mathcal{G}}(v_i)$ with $j > i$ **do**
    for each vertex $v_k \in Adj_{\mathcal{G}}(v_j)$ with $k > j$ **do**
      $\pi_2[v_k] \leftarrow v_j$
    for each vertex $v_k \in Adj_{\mathcal{G}}(v_j)$ with $k > j$ **do**
      if $\pi_1[v_k] = v_i$ **then**
        if $\sum_{e \in \binom{\{v_i, v_j, v_k\}}{2}}$ cover$[e] < 2$ **then**
          for each vertex $v_\ell \in Adj_{\mathcal{G}}(v_k)$ with $\ell > k$ **do**
            if $\pi_1[v_\ell] = v_i$ **and** $\pi_2[v_\ell] = v_j$ **then**
              if $\sum_{e \in \binom{\{v_i, v_j, v_k, v_\ell\}}{2}}$ cover$[e] < 4$ **then**
                $\mathcal{C}^4 \leftarrow \mathcal{C}^4 \cup \{\{v_i, v_j, v_k, v_\ell\}\}$ ▷ Add the 4-clique to $\mathcal{C}^4$
                for each edge $e \in \binom{\{v_i, v_j, v_k, v_\ell\}}{2}$ **do**
                  cover$[e] \leftarrow 1$
        if $\sum_{e \in \binom{\{v_i, v_j, v_k\}}{2}}$ cover$[e] < 2$ **then**
          $\mathcal{C}^4 \leftarrow \mathcal{C}^4 \cup \{\{v_i, v_j, v_k\}\}$ ▷ Add the 3-clique to $\mathcal{C}^4$
          for each edge $e \in \binom{\{v_i, v_j, v_k\}}{2}$ **do**
            cover$[e] \leftarrow 1$
      if cover$[e_{ij}] = 0$ **then**
        $\mathcal{C}^4 \leftarrow \mathcal{C}^4 \cup \{\{v_i, v_j\}\}$ ▷ Add the 2-clique to $\mathcal{C}^4$
        cover$[e_{ij}] \leftarrow 1$

ECC $\mathcal{C}$ of $\mathrm{G}(M)$. So, matrix A can be considered as a clique incidence matrix of $\mathrm{G}(M)$. That is, column c_h of matrix A corresponds to a clique $\mathcal{C}_h$ of $\mathcal{C}$, where $a_{ih} \neq 0$ implies that vertex $v_i \in \mathcal{C}_h$. The row-net hypergraph model $\mathrm{H}_{\mathrm{RN}}(A)$ of matrix A is equivalent to the clique-node hypergraph of graph $\mathrm{G}(M)$ for the ECC $\mathcal{C}$ determined by the columns of A, i.e., $\mathrm{H}_{\mathrm{RN}}(A) \equiv \mathrm{CNH}(\mathrm{G}(M), \mathcal{C})$. In other words, the NIG representation of row-net hypergraph model $\mathrm{H}_{\mathrm{RN}}(A)$ of matrix A is equivalent to $\mathrm{G}(M)$, i.e., $\mathrm{NIG}(\mathrm{H}_{\mathrm{RN}}(A)) \equiv \mathrm{G}(M)$.

As shown in [8], a K-way node-partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$, which induces a $(K+1)$-way net partition $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$, of $\mathrm{H}_{\mathrm{RN}}(A)$ can be decoded as permuting matrix A into a K-way rowwise singly-bordered block diagonal (SB) form

$$A_{SB} = PAQ = \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_K \\ A_{B_1} & \ldots & A_{B_K} \end{bmatrix}. \tag{3.1}$$

Here, the K-way node partition is used to define the partial column permutation matrix Q by permuting the columns corresponding to the nodes of part $\mathcal{U}_k$ after those corresponding to the nodes of part $\mathcal{U}_{k-1}$ for $2 \leq k \leq K$. The $(K+1)$-way partition on the nets of $\mathrm{H}_{\mathrm{RN}}(A)$ is used to define the partial row permutation matrix P by permuting the rows corresponding to the nets of $\mathcal{N}_k$ after those corresponding to the nets of $\mathcal{N}_{k-1}$ for $2 \leq k \leq K$, and permuting



(a) (b)

FIG. 3.4. *(a) Matrix A whose row-net hypergraph representation is given in Fig. 3.1(b) and its 3-way SB form A_{SB} induced by the 3-way partition Π_{HP} given in Fig. 3.2(a); (b) Matrix M whose standard graph representation is given in Fig. 3.1(a) and its 3-way DB form M_{DB} induced by A_{SB}*

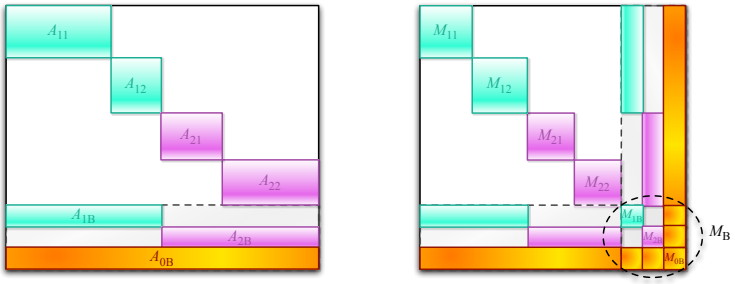
the rows corresponding to the external nets to the end. Here, the partitioning objective of minimizing the cutsize of Π_{HP} corresponds to minimizing the number of coupling rows in A_{SB}. The partitioning constraint of balancing on the internal net counts of node parts of Π_{HP} infers balance among the row counts of the rectangular diagonal submatrices in A_{SB}. It is clear that the transpose of A_{SB} will be in a columnwise SB form.

An SB form A_{SB} of A induces a DB form M_{DB} of M, since multiplying A_{SB} with its transpose produces a DB form of M. That is,

$$
\begin{aligned}
A_{SB}A_{SB}^T &= \begin{bmatrix} A_1 & & \\ & \ddots & \\ & & A_K \\ A_{B_1} & \dots & A_{B_K} \end{bmatrix} \begin{bmatrix} A_1^T & & & A_{B_1}^T \\ & \ddots & & \vdots \\ & & A_K^T & A_{B_K}^T \end{bmatrix} \\
&= \begin{bmatrix} A_1A_1^T & & & A_1A_{B_1}^T \\ & \ddots & & \vdots \\ & & A_KA_K^T & A_KA_{B_1}^T \\ A_{B_1}A_1^T & \dots & A_{B_K}A_K^T & \sum_k A_{B_k}A_k^T \end{bmatrix} = M_{DB}
\end{aligned} \tag{3.2}
$$

As seen in (3.2), the number of rows/columns in the square diagonal block $A_kA_k^T$ of M_{DB} is equal to the number of rows of the rectangular diagonal block A_k of A_{SB}. Furthermore, the number of coupling rows/columns in M_{DB} is equal to the number of coupling rows in A_{SB}. So, minimizing the number of coupling rows in A_{SB} corresponds to minimizing the number of coupling rows/columns in M_{DB}, whereas balancing on row counts of the rectangular diagonal submatrices in A_{SB} infers balance among the row/column counts of the square diagonal submatrices in M_{DB}. Thus, given a structural factorization $M = AA^T$ of matrix M, the proposed HP-based GPVS formulation corresponds to formulating the problem of permuting M into a DB block diagonal form as an instance of the problem of permuting A into an SB block diagonal form. Figure 3.4 shows the matrix theoretical view of our HP-based GPVS formulation on the sample graph, hypergraph and their partitions given in Figures 3.1 and 3.2.

4. HP-based fill-reducing ordering. Given a $p \times p$ symmetric and square matrix $M = \{m_{ij}\}$ for fill reducing ordering, let $\mathcal{G}(M) = (\mathcal{V}, \mathcal{E})$ denote the standard graph representation of matrix M.



(a) (b)

FIG. 4.1. *(a) A sample 4-way SB form of a matrix A obtained through a 2-level recursive hypergraph bipartitioning process; (b) the corresponding 4-way DB form of matrix $M = AA^T$*

4.1. Incomplete-nested-dissection-based orderings via recursive hypergraph bipartitioning. As described in [6], the fill-reducing matrix reordering schemes based on incomplete nested dissection can be classified as: *nested dissection* (ND) and *multisection* (MS). Both schemes apply 2-way GPVS (bisection) recursively on $\mathrm{G}(M)$ until the parts (domains) become fairly small. After each bisection step, the vertices in the 2-way separator (bisector) are removed and the further bisection operations are recursively performed on the subgraphs induced by the parts of bisection. In the proposed recursive-HP-based ordering approach, the constructed hypergraph $\mathcal{H}$ (where $\mathrm{NIG}(\mathcal{H}) \equiv \mathrm{G}(M)$) is bipartitioned recursively until the number of internal nets of the parts become fairly small. After each bipartitioning step, the cut nets are removed and the further bipartitioning operations are recursively performed on the sub-hypergraphs induced by the node parts of the bipartition. Note that this cut-net removal scheme in recursive 2-way HP corresponds to the above-mentioned separator-vertex removal scheme in recursive 2-way GPVS.

As mentioned above, both ND and MS schemes effectively obtain a multiway separator (multisector) at the end of the recursive 2-way GPVS operations. In both schemes, the parts of the multiway separator are ordered using an MD-based algorithm before the separator. It is clear that the parts can be ordered independently. These two schemes differ in the order that they number the vertices of the multiway separator. In the ND scheme, the 2-way separators constituting the multiway separator are numbered using an MD-based algorithm in depth-first order of the recursive bisection process. Note that the 2-way separators at the same level of the recursive bisection tree can be ordered independently. In the MS scheme, the multiway separator is ordered using an MD-based algorithm as a whole in a single step.

Figure 4.1 displays a sample 4-way SB form of matrix A and the corresponding 4-way DB form of matrix M induced by a 2-level recursive bipartitioning/bisection process. Here, the bipartitioning/bisection operation at the root level is numbered as 0, whereas the bipartitioning/bisection operations at the second level are numbered as 1 and 2. The parts of a bipartition/bisection are always numbered as 1 and 2, whereas the border is numbered as B. For example, A_{11}/M_{11} and A_{12}/M_{12} denote the diagonal domain submatrices corresponding to the two parts of the bipartitioning/bisection operation 1, whereas A_{21}/M_{21} and A_{22}/M_{22} denote the diagonal domain submatrices corresponding to the two parts of the bipartitioning/bisection operation 2. As seen in the figure, $M_{0B} = A_{0B}A_{0B}^T$ denotes the diagonal border submatrix corresponding to the 2-way separator obtained at the root level, whereas $M_{1B} = A_{1B}A_{1B}^T$ and $M_{2B} = A_{2B}A_{2B}^T$ denote the diagonal border submatrices corresponding to the 2-way separators obtained at the second level. Note that M_B denotes the diagonal border submatrix corresponding to the overall 4-way separator. In both ND and

MS schemes, diagonal domain submatrices are ordered before the diagonal border submatrix M_B. In the ND scheme, the Schur complements of the diagonal border submatrices are ordered in depth-first order M_{1B}, M_{2B} and M_{0B} of the recursive bisection process. In the MS scheme, the Schur complement of the overall diagonal border submatrix M_B is ordered as a whole.

4.2. Structural factor sparsening for ordering LP matrices. Interior point methods are widely used for solving linear programming problems [20]. These are iterative methods and usually adopt the normal equations approach [3]. The main computational cost at each iteration is the solution of a symmetric positive definite system of the form $Mx = b$, where $M = AD^2A^T$. Here, $A = \{a_{ij}\}$ is a $p \times q$ sparse rectangular constraint matrix which remains constant throughout the iterations, and D^2 is a $q \times q$ diagonal scaling matrix which changes from iteration to iteration. This linear system is typically solved by computing the Cholesky factorization ($M = LL^T$) of M, and solving the triangular system through forward and backward substitution. So, fill-reducing ordering of matrix M is crucial in the overall performance of the interior point algorithm.

Since D^2 is a diagonal matrix, AA^T determines the sparsity pattern of M. So, by neglecting numerical cancellations that may occur in matrix-matrix-transpose multiplication AA^T, we can consider $A = \{a_{ij}\}$ as a {0,1}-matrix so that $M = AA^T$ gives us a structural factorization of matrix M. Note that matrix A may contain redundant columns and/or nonzeros in terms of determining the sparsity pattern of M. Here, we will propose and discuss two matrix sparsening algorithms which aim at deleting as many columns and/or nonzeros of matrix A without disturbing the sparsity pattern of matrix M. The objective is to speed up the proposed HP-based GPVS method for ordering LP matrices through decreasing the size of the row-net hypergraph representation of matrix A. Both algorithms consider both column and nonzero deletions. However the first algorithm is nonzero-deletion oriented, whereas the second one is column-deletion oriented.

4.2.1. Nonzero-deletion-oriented sparsening. We define b_{ij} to denote the number of common columns between rows r_i and r_j of matrix A. A column c_h is said to be common between rows r_i and r_j if both rows have a nonzero in column c_h. Note that b_{ij} is equal to the integer value of nonzero m_{ij} of matrix M if $M = AA^T$ is computed using A as a {0,1}-matrix. So, the sparsity pattern of M will remain to be the same as long as b_{ij} values corresponding to the nonzeros of matrix M remain to be greater than or equal to 1 during nonzero deletions in matrix A. In particular, a nonzero a_{ih} of matrix A can be deleted if $b_{ij} > 1$ for each nonzero a_{jh} in column c_h of matrix A.

The proposed nonzero-deletion-based sparsening algorithm is given in Algorithm 3. The algorithm does not require M matrix as input. Note that the quality of the sparsening depends on processing order of nonzeros for deletion. Algorithm 3 considers the nonzeros for deletion in row major order. In the doubly-nested for loop in lines 4 – 6, the b_{ij} values for row r_i are computed in 1D array B. Then, for each nonzero a_{ih} in row r_i, the for loop in lines 9 – 12 checks whether the condition $b_{ij} > 1$ holds for each nonzero a_{jh} in column c_h of matrix A. If it is so, the relevant b_{ij} (i.e., B_j) values are decremented and the nonzero a_{ih} is deleted in lines 13 – 16. At the end of the algorithm, the columns that become empty due to the nonzero deletions are detected and deleted by the for loop in lines 20 – 22. This algorithm runs in $O(\sum_{c_h \in A} |c_h|^2)$ time, where $|c_h|$ denotes the number of nonzeros in column c_h.

4.2.2. Column-deletion-oriented sparsening. Consider the problem of maximizing the number of A-matrix column deletions without disturbing the sparsity pattern of matrix M. This problem can be formulated as a minimum set cover problem as follows: The set of M-matrix nonzeros constitutes the main set of elements, whereas the set of A-matrix

Algorithm 3 Nonzero-Deletion-Oriented Sparsening Algorithm

Require: A: both in CSR and CSC formats
1: **for each** row $r_i \in A$ **do**
2: $B[i] \leftarrow 0$
3: **for each** row $r_i \in A$ **do**
4: **for each** nonzero $a_{ih} \in r_i$ **do**
5: **for each** nonzero $a_{jh} \in c_h$ **do**
6: $B[j] \leftarrow B[j]+1$
7: **for each** nonzero $a_{ih} \in r_i$ **do**
8: $flag \leftarrow$ TRUE
9: **for each** nonzero $a_{jh} \in c_h$ **do**
10: **if** $B[j] = 1$ **then**
11: $flag \leftarrow$ FALSE
12: **break**
13: **if** $flag =$ TRUE **then**
14: **for each** nonzero $a_{jh} \in c_h$ **do**
15: $B[j] \leftarrow B[j]-1$
16: delete nonzero a_{ih}
17: **for each** nonzero $a_{ih} \in r_i$ **do**
18: **for each** nonzero $a_{jh} \in c_h$ **do**
19: $B[j] \leftarrow 0$
20: **for each** column $c_h \in A$ **do**
21: **if** c_h is empty **then**
22: delete column c_h

columns constitutes the family $\mathcal{F}$ of subsets of main set. For each A-matrix column c_h, the subset $\mathrm{S}(c_h)$ of main set of elements is defined as $\mathrm{S}(c_h) = \{m_{ij} \in M : a_{ih}$ and a_{jh} are nonzeros$\}$. That is, each nonzero pair (a_{ih}, a_{jh}) in column c_h contributes m_{ij} to the subset $\mathrm{S}(c_h)$. A subset of $\mathcal{F}$ is said to cover the main set of elements if its union is equal to the main set. The objective of the minimum set cover problem is to find a minimum number of subsets covering the main set. This objective corresponds to minimizing the number of A-matrix columns to be retained (maximizing the number of A-matrix columns to be deleted) without disturbing the sparsity pattern of matrix M.

The minimum set cover problem is known to be NP-hard [34]. However, there is a well known $(\ln n)$-approximation algorithm [19], which works as follows: It grows a cover set using a sequence of greedy decisions. The greedy decision at each step is to select a subset that covers as many uncovered elements as possible. The algorithm terminates when all elements are covered.

A two-phase sparsening algorithm can be developed based on this minimum set cover algorithm as follows: In the first phase, the set cover algorithm is used to obtain a matrix A_c whose columns correspond to a minimum set of A-matrix columns that covers the set of all nonzeros of M. In the second phase, Algorithm 3 is run on matrix A_c for nonzero deletions. However, for the sake of efficiency, we propose Algorithm 4 which interleaves column selection operations with nonzero deletion operations.

In Algorithm 4, two successive for loops at lines 2 – 10 construct the list of A-matrix columns that cover each M-matrix nonzero and compute the number of M-matrix nonzeros covered by each A-matrix column. A priority queue $\mathcal{Q}$ which contains all A-matrix columns is built in line 11, where A-matrix columns are keyed by the number of uncovered M-matrix nonzeros that they cover. The while loop in lines 12 – 26 repeatedly identifies and selects an A-matrix column that covers the maximum number of uncovered M-matrix nonzeros and then updates the relevant data structures accordingly. Meanwhile, the nonzeros of the selected

Algorithm 4 Column-Deletion-Oriented Sparsening Algorithm

Require: A: in CSC format, M: in CSR / CSC format
1: totalUncovered $\leftarrow 0$
2: **for each** nonzero $m_{ij} \in M$ with $i \leq j$ **do**
3: ColCoverList[m_{ij}] $\leftarrow \emptyset$
4: covered[m_{ij}] $\leftarrow$ FALSE
5: totalUncovered $\leftarrow$ totalUncovered $+1$
6: **for each** column $c_h \in A$ **do**
7: key[c_h] $\leftarrow 0$
8: **for each** nonzero pair $(a_{ih}, a_{jh}) \in c_h$ with $i \leq j$ **do**
9: ColCoverList[m_{ij}] $\leftarrow$ ColCoverList[m_{ij}] $\cup \{c_h\}$
10: key[c_h] $\leftarrow$ key[c_h] $+1$
11: $\mathcal{Q} \leftarrow$ PriorityQueue($\{c_h : c_h$ is column of $A\}$, key)
12: **while** totalUncovered > 0 **do**
13: $c_h \leftarrow$ EXTRACT-MAX($\mathcal{Q}$)
14: **for each** nonzero $a_{ih} \in c_h$ **do**
15: DeleteFlag[a_{ih}] $\leftarrow$ TRUE
16: **for each** nonzero pair $(a_{ih}, a_{jh}) \in c_h$ with $i \leq j$ **do**
17: ColCoverList[m_{ij}] $\leftarrow$ ColCoverList[m_{ij}] $-\{c_h\}$
18: **if** covered[m_{ij}] $=$ FALSE **then**
19: covered[m_{ij}] $\leftarrow$ TRUE
20: totalUncovered $\leftarrow$ totalUncovered -1
21: **for each** column $c_{h'} \in$ ColCoverList[m_{ij}] **do**
22: key[$c_{h'}$] $\leftarrow$ key[$c_{h'}$] -1 ▷ DECREASE-KEY operation
23: DeleteFlag[a_{ih}] $\leftarrow$ DeleteFlag[a_{jh}] $\leftarrow$ FALSE
24: **for each** nonzero $a_{ih} \in c_h$ **do**
25: **if** DeleteFlag[a_{ih}] $=$ TRUE **then**
26: delete nonzero a_{ih}
27: **for each** column $c_h \in \mathcal{Q}$ **do**
28: delete column c_h

A-matrix column are processed for deletion with respect to set of previously selected and processed A-matrix columns. After detecting a full coverage of the M-matrix nonzeros, the set of columns remaining in the priority queue are deleted in lines 27 – 28. This algorithm can also be made to run in $O(\sum_{c_h \in A} |c_h|^2)$ time through a priority queue implementation with $O(1)$–time extract-max and decrease-key operations. A sorted linear array implementation can achieve the desired bounds for those operations by exploiting the bounded integer key values and decrement-type decrease-key operations.

5. Experimental Results. The proposed HP-based GPVS formulation is embedded into the state-of-the-art HP tool PaToH [14] and the resulting HP-based fill-reducing ordering tool is referred to here as oPaToH. In oPaToH, the recursive hypergraph bipartitioning process is continued until the number of internal nets of a part of a bipartition drops below 200 or the number of nodes of a part of a bipartition drops below 100. oPaToH implements both MS and ND schemes which will be referred as oPaToH-MS and oPaToH-ND, respectively. Both oPaToH-MS and oPaToH-ND use the CMD algorithm for ordering decoupled diagonal domain submatrices and the MMD algorithm for ordering Schur complements of the diagonal border submatrices.

The performance of oPaToH is compared against the state-of-the-art ordering algorithms and tools MeTiS [37], MMD [40] implementation from SPARSPAK [23], and SMOOTH [5][1].

[1]SMOOTH sparse matrix ordering package later included in sparse linear system solver package called SPOOLES [4]. We will continue to use name SMOOTH to denote that we are referring the ordering package.

MeTiS v4.0 [37] provides two multilevel nested dissection [36] programs, one is GPVS based, onmetis, the other, oemetis, is GPES based. Both onmetis and oemetis use MMD for ordering decoupled diagonal domain submatrices and Schur complements of the diagonal border submatrices. We use SMOOTH with MS scheme, CMD for ordering decoupled diagonal domain submatrices and MMD for ordering Schur complements of the diagonal border submatrices. All ordering tools and methods were run on a PC equipped with a 1.6 Ghz Pentium 4, and 1 GB memory.

We performed experimental evaluation of the proposed HP-based fill reducing ordering approach using 50 matrices obtained from the University of Florida sparse matrix collection [21]. The first 25 matrices are general symmetric and square M matrices arising in different application domains, whereas the remaining 25 M matrices are derived from LP constraint matrices using $M = AA^T$. Table 5.1 illustrates the properties of these matrices. In this table, p and $nnz(M)$ denote, respectively, the number of rows/columns and nonzeros of matrix M. For an M-matrix derived from an LP problems, the number of columns q and nonzeros $nnz(A)$ are also listed for the respective A-matrix. Note that the number of rows of A is equal to the number of rows/columns of M. The general matrices are further divided into three groups (first 5, second 5 and remaining 15) according to the size of the maximum cliques that can be obtained from their graph representations. The reason for this division will become clear during the discussion of Table 5.3. The matrices in each category/group are listed in increasing order of number of nonzeros. This table also displays the performance of the onmetis ordering in terms of operation count in triangular factorization (shown as "opc"), number of nonzeros in the triangular factor (shown as $nnz(L)$), and ordering time in seconds.

Table 5.2 compares the performance of nonzero-deletion-oriented (Alg3) and column-deletion-oriented (Alg4) matrix sparsening algorithms for ordering LP matrices. In the first 4 columns of Table 5.2, the sparsening performances of these two algorithms are compared in terms of the number of remaining columns and nonzeros after deletion, which are normalized with respect to the number of columns and nonzeros of the original A matrix, respectively. From now on, $\tilde{A}$ and $\tilde{q}$ refer to the sparsened A matrix and its number of columns. As seen in the table, Algorithm 3 and Algorithm 4 display very close sparsening performance in terms of both number of columns and nonzeros. In the last 6 columns of Table 5.2, the ordering quality and run-time performances obtained by oPaToH-MS using these two sparsening algorithms are displayed as normalized with respect to those obtained by oPaToH-MS using the original A matrix without sparsening. Here, total ordering time includes A-matrix sparsening, hypergraph construction, hypergraph partitioning and ordering times. As seen in the table, oPaToH-MS using Algorithm 3 and Algorithm 4 displays very close performance in ordering quality in terms of both operation-count and fill-in metrics. As seen in the table, the sparsened A-matrices obtained by both algorithms do not degrade the ordering quality of oPaToH-MS. In fact the sparsened A-matrices obtained by Algorithm 3 lead to slight improvements in the ordering quality (e.g., 2% less operation count on the average). As also seen in the table, both sparsening algorithms amortize by reducing the total ordering time of oPaToH-MS considerably. In the relative comparison of these two algorithms, oPaToH-MS using Algorithm 3 runs considerably faster than oPaToH-MS using Algorithm 4 in almost all ordering instances (except one instance). oPaToH-MS using Algorithm 3 and Algorithm 4 for sparsening respectively runs 20% and 8% faster than oPaToH-MS using the original A matrix, on the average. Very similar results were obtained in the relative performances of these two sparsening algorithms in oPaToH-ND. Therefore, Algorithm 3 is used for sparsening in oPaToH for LP matrices.

Table 5.3 displays the properties of the hypergraphs in terms of number of nodes and

TABLE 5.1
Properties of test matrices and results of onmetis orderings

	$p \times p$ M matrix		$p \times q$ A matrix		onmetis		
name	p	$nnz(M)$	q	$nnz(A)$	opc	$nnz(L)$	time (s)
General M matrices							
ncvxqp9	23,047	45,540	-	-	5.94E+06	123,462	0.16
aug3dcqp	50,286	100,572	-	-	2.78E+08	1,011,206	0.54
c-53	170,989	341,978	-	-	3.03E+07	391,804	0.57
c-59	219,627	439,254	-	-	2.48E+09	3,221,503	0.96
c-67	236,980	473,960	-	-	1.32E+07	439,898	1.06
lshp3025	8,904	17,808	-	-	3.01E+06	72,058	0.01
lshp3466	10,215	20,430	-	-	3.65E+06	84,338	0.02
bodyy4	52,196	104,392	-	-	3.29E+07	501,494	0.18
rail_20209	59,512	119,024	-	-	1.31E+07	319,401	0.24
cvxbqp1	149,984	299,968	-	-	4.56E+08	1,978,965	0.65
shuttle_eddy	46,585	93,170	-	-	2.07E+07	352,363	0.11
nasa4704	50,026	100,052	-	-	3.88E+07	303,720	0.03
bcsstk24	78,174	156,348	-	-	4.14E+07	314,104	0.02
skirt	91,964	183,925	-	-	2.92E+07	465,553	0.17
bcsstk28	107,307	214,614	-	-	5.40E+07	403,052	0.03
s1rmq4m1	137,811	275,622	-	-	1.07E+08	646,878	0.03
vibrobox	165,250	330,500	-	-	1.31E+09	2,482,629	0.34
crystk01	155,508	311,016	-	-	2.73E+08	1,003,272	0.60
bcsstm36	165,097	319,314	-	-	1.14E+08	879,713	0.38
gridgena	231,561	463,122	-	-	3.61E+08	2,671,255	0.72
k1_san	256,411	512,821	-	-	4.00E+08	2,605,291	1.20
finan512	261,120	522,240	-	-	1.56E+08	1,747,840	1.29
msc23052	565,881	1,131,762	-	-	6.40E+08	2,934,092	0.22
bcsstk35	709,963	1,419,926	-	-	4.81E+08	3,049,203	0.30
oilpan	1,761,718	3,523,436	-	-	2.78E+09	9,137,443	0.64
Linear programming $M = AA^T$ matrices							
delf_A_36	3,170	33,508	6,654	15,397	1.76E+006	50,185	0.03
lp_dfl001	6,071	82,267	12,230	35,632	7.34E+008	1,277,948	0.12
model9	2,879	103,961	10,939	55,956	4.86E+006	97,041	0.06
nl	7,039	105,089	15,325	47,035	4.20E+007	298,181	0.09
ge	10,099	112,129	16,369	44,825	2.28E+007	267,891	0.14
lp_ken_13	28,632	161,804	42,659	97,246	1.72E+007	349,677	0.30
lpi_gosh	3,792	206,010	13,455	99,953	3.66E+007	249,770	0.11
cq9	9,278	221,590	21,534	96,653	4.10E+007	406,070	0.17
lp_osa_14	2,337	230,023	54,797	317,097	6.21E+006	116,160	0.14
co9	10,789	249,205	22,924	109,651	5.03E+007	471,887	0.20
pltexpa	26,894	269,736	70,364	143,059	1.57E+008	1,229,360	0.50
model10	4,400	293,260	16,819	150,372	5.69E+007	392,002	0.15
fome12	24,284	329,068	48,920	142,528	2.58E+009	4,732,776	0.67
lp_cre_d	8,926	372,266	73,948	246,614	2.05E+008	752,413	0.35
r05	5,190	406,158	9,690	104,145	1.21E+008	528,707	0.17
world	34,506	582,064	67,147	198,883	3.52E+008	2,001,206	0.83
mod2	34,774	604,910	66,409	199,810	4.24E+008	2,214,268	0.87
lp_maros_r7	3,136	664,080	9,408	144,848	7.19E+008	1,401,207	0.32
ex3sta1	17,443	679,857	17,516	68,779	7.89E+009	8,052,424	0.45
psse2	28,634	736,338	11,028	115,262	7.67E+007	963,141	0.22
fxm3_16	41,340	765,526	85,575	392,252	2.76E+007	686,497	0.90
Kemelmacher	28,452	781,148	9,693	100,875	1.11E+009	4,152,721	0.94
graphics	29,493	1,577,187	11,822	117,954	1.33E+009	4,966,956	0.39
stat96v5	2,307	1,790,467	75,779	233,921	2.55E+009	2,169,949	0.67

TABLE 5.2
Sparsening properties and ordering performances of sparsening algorithms relative to original A matrix for LP problems

	Sparsening properties				Ordering performance of oPaToH-MS					
	Alg3		Alg4		*opc*		$nnz(L)$		time	
name	$\tilde{q}$	$nnz(\tilde{A})$	$\tilde{q}$	$nnz(\tilde{A})$	Alg3	Alg4	Alg3	Alg4	Alg3	Alg4
lp_pds_02	0.98	0.99	0.98	0.99	0.93	0.99	0.98	1.00	0.95	0.92
delf_A_36	0.41	0.61	0.38	0.59	0.92	1.10	0.98	1.02	0.77	0.83
lp_dfl001	0.87	0.94	0.87	0.94	1.01	0.99	1.00	0.99	0.97	1.10
model9	0.59	0.86	0.60	0.86	1.02	1.00	1.00	1.00	0.99	1.09
nl	0.53	0.77	0.52	0.76	0.95	0.97	0.99	0.99	1.04	1.12
ge	0.57	0.71	0.56	0.71	0.94	0.87	0.99	0.97	0.88	0.92
lp_ken_13	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.06	1.15
lpi_gosh	0.73	0.95	0.73	0.95	0.96	0.99	0.99	0.99	0.98	1.08
cq9	0.54	0.73	0.53	0.72	1.01	1.02	1.00	1.01	0.76	0.88
lp_osa_14	0.96	0.52	0.96	0.52	1.00	1.00	1.00	1.00	0.12	0.17
co9	0.54	0.72	0.52	0.71	1.00	1.00	1.00	1.00	0.70	0.80
pltexpa	0.57	0.79	0.57	0.79	1.00	1.12	1.00	1.03	1.15	1.19
model10	0.86	0.94	0.86	0.94	0.95	0.97	0.98	0.98	0.97	1.11
fome12	0.87	0.94	0.87	0.94	1.01	1.02	1.00	1.01	1.01	1.08
lp_cre_d	0.94	0.86	0.97	0.87	0.93	0.98	0.97	0.99	0.78	0.85
r05	0.93	0.99	0.93	0.99	1.00	1.00	1.00	1.00	1.30	1.63
world	0.45	0.79	0.45	0.79	0.99	1.00	1.00	1.00	0.97	0.97
mod2	0.45	0.79	0.45	0.79	1.01	1.01	1.00	1.00	0.90	0.98
lp_maros_r7	0.32	0.50	0.32	0.44	0.99	1.00	1.00	1.00	0.88	1.21
ex3sta1	0.41	0.82	0.41	0.82	1.03	1.01	1.01	1.00	1.08	1.21
psse2	0.43	0.51	0.43	0.51	1.01	1.05	1.00	1.01	0.91	1.16
fxm3_16	0.56	0.53	0.55	0.51	0.98	0.98	1.00	0.99	0.63	0.66
Kemelmacher	0.99	1.00	0.99	1.00	0.97	1.00	0.99	1.00	0.98	1.13
graphics	0.33	0.75	0.33	0.75	0.99	1.00	1.00	1.00	1.21	1.68
stat96v5	0.02	0.04	0.02	0.04	0.93	0.93	0.97	0.97	0.13	0.23
geomean	**0.53**	**0.69**	**0.53**	**0.68**	**0.98**	**1.00**	**0.99**	**1.00**	**0.80**	**0.92**

pins. In the table, $\mathcal{H}^2$, $\mathcal{H}^3$ and $\mathcal{H}^4$ denote the clique-node hypergraphs induced by ECCs $\mathcal{C}^2$, $\mathcal{C}^3$ and $\mathcal{C}^4$, respectively. Recall that ECCs $\mathcal{C}^3$ and $\mathcal{C}^4$ are constructed from $\mathrm{G}(M)$ using Algorithm 1 and Algorithm 2, respectively. For LP matrices, $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ refers to the hypergraphs obtained from row-net representations of the sparsened A matrices. Note that, for ordering LP matrices, we recommend to use $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ hypergraphs. Here, we provide the results for $\mathcal{H}^2$, $\mathcal{H}^3$ and $\mathcal{H}^4$ hypergraphs are given for the sake of completeness. Also note that, for a given M matrix, all hypergraphs have the same number of nets, which is equal to the number of rows/columns of M. In the table, the $\mathcal{H}^2$ model is considered as the base model, so the number of nodes and pins of $\mathcal{H}^3$, $\mathcal{H}^4$ and $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ are displayed as normalized with respect to those of $\mathcal{H}^2$.

As seen in Table 5.3, the size of clique-node hypergraph for a given M matrix decreases in terms of both number of nodes and pins when larger cliques of $\mathrm{G}(M)$ are considered while constructing the hypergraph. That is, $\mathcal{H}^4$ has smaller size than $\mathcal{H}^3$, which in turn has smaller size than $\mathcal{H}^2$. However, the first five and the first ten out of 25 general matrices do not lead to 3-cliques and 4-cliques, respectively. So, the $\mathcal{H}^2$, $\mathcal{H}^3$ and $\mathcal{H}^4$ hypergraphs are the same for the first five general M matrices, whereas the $\mathcal{H}^3$ and $\mathcal{H}^4$ hypergraphs are the same for the first ten general M matrices. The second `geomean` row shows the values when those five and ten matrices are excluded from the geometric averaging. That is, 6– and 11– refer to the geometric averages when the first five and the first ten matrices are excluded, respectively. As seen in Table 5.3, for LP matrices, $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ hypergraphs have drastically smaller size than even $\mathcal{H}^4$ hypergraphs in general.

TABLE 5.3
Hypergraph properties

		$\mathcal{H}^2$		$\mathcal{H}^3$		$\mathcal{H}^4$		$\mathrm{H}_{\mathrm{RN}}(\tilde{A})$	
name	$\#nets$	$\#nodes$	$\#pins$	$\#nodes$	$\#pins$	$\#nodes$	$\#pins$	$\#nodes$	$\#pins$
General Matrices									
ncvxqp9	23,047	23,047	45,540	1.00	1.00	1.00	1.00	-	-
aug3dcqp	50,286	50,286	100,572	1.00	1.00	1.00	1.00	-	-
c-53	170,989	170,989	341,978	1.00	1.00	1.00	1.00	-	-
c-59	219,627	219,627	439,254	1.00	1.00	1.00	1.00	-	-
c-67	236,980	236,980	473,960	1.00	1.00	1.00	1.00	-	-
lshp3025	8,904	8,904	17,808	0.35	0.53	0.35	0.53	-	-
lshp3466	10,215	10,215	20,430	0.35	0.53	0.35	0.53	-	-
bodyy4	52,196	52,196	104,392	0.36	0.53	0.36	0.53	-	-
rail_20209	59,512	59,512	119,024	0.48	0.71	0.48	0.71	-	-
cvxbqp1	149,984	149,984	299,968	0.45	0.67	0.45	0.67	-	-
shuttle_eddy	46,585	46,585	93,170	0.51	0.75	0.43	0.68	-	-
nasa4704	50,026	50,026	100,052	0.48	0.72	0.29	0.57	-	-
bcsstk24	78,174	78,174	156,348	0.49	0.73	0.30	0.60	-	-
skirt	91,964	91,964	183,925	0.48	0.72	0.30	0.57	-	-
bcsstk28	107,307	107,307	214,614	0.49	0.73	0.31	0.62	-	-
s1rmq4m1	137,811	137,811	275,622	0.49	0.74	0.32	0.64	-	-
vibrobox	165,250	165,250	330,500	0.50	0.74	0.33	0.61	-	-
crystk01	155,508	155,508	311,016	0.50	0.74	0.31	0.62	-	-
bcsstm36	165,097	165,097	319,314	0.52	0.74	0.34	0.59	-	-
gridgena	231,561	231,561	463,122	0.54	0.74	0.39	0.60	-	-
k1_san	256,411	256,411	512,821	0.45	0.65	0.27	0.49	-	-
finan512	261,120	261,120	522,240	0.49	0.68	0.27	0.47	-	-
msc23052	565,881	565,881	1,131,762	0.49	0.74	0.32	0.63	-	-
bcsstk35	709,963	709,963	1,419,926	0.49	0.73	0.30	0.60	-	-
oilpan	1,761,718	1,761,718	3,523,436	0.49	0.74	0.30	0.59	-	-
geomean				**0.54**	**0.74**	**0.42**	**0.65**	**-**	**-**
6–,11–				**0.47**	**0.69**	**0.32**	**0.59**	**-**	
LP Problems									
lp_pds_02	10,164	2,953	20,328	0.75	0.83	0.74	0.81	0.74	0.81
delf_A_36	15,169	3,170	30,338	0.48	0.70	0.34	0.60	0.18	0.31
lp_dfl001	38,098	6,071	76,196	0.51	0.70	0.37	0.56	0.28	0.44
model9	50,730	2,879	101,271	0.51	0.75	0.33	0.62	0.13	0.48
nl	49,034	7,039	98,059	0.52	0.74	0.36	0.61	0.16	0.37
ge	51,015	10,099	102,030	0.50	0.72	0.35	0.60	0.18	0.31
lp_ken_13	66,586	28,632	133,172	0.62	0.72	0.62	0.72	0.64	0.73
lpi_gosh	101,213	3,792	202,322	0.52	0.75	0.35	0.66	0.10	0.47
cq9	106,187	9,278	212,343	0.50	0.74	0.34	0.60	0.11	0.33
lp_osa_14	113,843	2,337	227,686	0.51	0.76	0.48	0.74	0.46	0.73
co9	119,330	10,789	238,538	0.50	0.74	0.35	0.63	0.10	0.33
pltexpa	121,421	26,894	242,842	0.51	0.69	0.43	0.63	0.33	0.46
model10	144,431	4,400	288,861	0.51	0.75	0.33	0.62	0.10	0.49
fome12	152,392	24,284	304,784	0.51	0.70	0.37	0.56	0.28	0.44
lp_cre_d	184,120	8,926	365,790	0.57	0.78	0.47	0.68	0.38	0.58
r05	200,503	5,190	400,987	0.50	0.74	0.34	0.66	0.04	0.26
world	274,179	34,506	547,958	0.49	0.72	0.34	0.60	0.11	0.29
mod2	285,487	34,774	570,555	0.49	0.72	0.34	0.60	0.10	0.28
lp_maros_r7	330,472	3,136	660,944	0.50	0.75	0.35	0.70	0.01	0.11
ex3sta1	331,207	17,443	662,414	0.49	0.73	0.31	0.61	0.02	0.08
psse2	353,852	28,634	707,704	0.48	0.72	0.30	0.60	0.01	0.08
fxm3_16	362,093	41,340	724,186	0.51	0.74	0.37	0.65	0.13	0.29
Kemelmacher	376,348	28,452	752,696	0.48	0.72	0.31	0.62	0.03	0.13
graphics	773,847	29,493	1,547,694	0.49	0.74	0.32	0.64	0.01	0.06
stat96v5	894,082	2,307	1,788,162	0.50	0.75	0.34	0.68	0.00	0.01
geomean				**0.52**	**0.74**	**0.37**	**0.64**	**0.09**	**0.26**

The performances of the ordering tools are displayed in Tables 5.4, 5.5 and 5.6 as normalized to those of onmetis. Tables 5.4 and 5.5 compare the ordering quality of the tools in terms of operation-count and fill-in metrics, respectively, whereas Table 5.6 compares the running times of the tools. In these tables, "oe" and "SM" abbreviations are used for oemetis and SMOOTH, respectively. For general matrices, in the second `geomean` row, 1–, 6– and 11– refer to the geometric averages when none, the first five and the first ten of the matrices are excluded, respectively.

First, we discuss the relative ordering quality performance of existing methods and tools on the results displayed in Tables 5.4 and 5.5. For general matrices, MMD shows the worst performance compared to the other ordering tools. SMOOTH and onmetis show very close performance both in terms of operation-count and fill-in metrics. oemetis also shows close performance to onmetis in terms of fill-in metric, however it performs considerably worse than (8% worse on average) onmetis in terms of operation-count metric. These results confirm that GPVS-based ordering in general performs better than GPES-based ordering, and comply with the results reported in [32]. Our results also show that, for LP matrices, onmetis performs drastically better than all other existing methods and tools, on the average. As seen in Table 5.6, MMD is the fastest for general matrices, whereas oemetis is the fastest for LP matrices, on the average. However, onmetis is the second fastest tool in both general and LP matrices, on the average.

Second, we discuss the effect of different clique cover finding algorithms and ordering schemes implemented in oPaToH. As seen in Tables 5.4 and 5.5, the ordering quality of oPaToH increases in general when larger cliques of $\mathrm{G}(M)$ are considered while constructing the hypergraph. That is, in general, oPaToH using $\mathcal{H}^4$ produces better orderings than oPaToH using $\mathcal{H}^3$, which in turn produces better orderings than oPaToH using $\mathcal{H}^2$. Note that the second `geomean` row should be considered while making this evaluation since using $\mathcal{H}^3/\mathcal{H}^4$ will not improve the performance if $\mathrm{G}(M)$ does not contain 3-cliques/4-cliques at all. For LP matrices, oPaToH using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ usually produces better orderings than oPaToH using $\mathcal{H}^2$, $\mathcal{H}^3$ and $\mathcal{H}^4$. These results justify our earlier choice on the use of $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ for ordering LP matrices. As seen in Tables 5.4 and 5.5, oPaToH-MS performs better than oPaToH-ND in terms of both operation-count and fill-in metrics. These experimental results comply with the results reported in [7] that favor the MS scheme over the ND scheme. Furthermore, as seen in Table 5.6, oPaToH-MS and oPaToH-ND shows very close performance in running time. So, we recommend the use of the MS scheme in our oPaToH ordering tool.

Third, we discuss the ordering performance of oPaToH-MS with respect to onmetis, since onmetis appears to be the best existing ordering tool, on the overall average. As seen in Tables 5.4 and 5.5, oPaToH produces considerably better orderings than onmetis, for both general and LP matrices, where the performance gap is more pronounced in the ordering of LP matrices. As seen in Table 5.4, the ordering quality of oPaToH-MS increases with increasing clique sizes used in clique-node hypergraph construction, on the average. For example, for general matrices, oPaToH-MS using $\mathcal{H}^2$,$\mathcal{H}^3$ and $\mathcal{H}^4$ produce orderings with 8%, 12% and 16% less operation count than onmetis, respectively, on the average. For LP matrices, oPaToH-MS using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ produces orderings with 21% less operation count than onmetis, on the average. Comparison of Tables 5.4 and 5.5 shows that the performance gap between oPaToH and onmetis is smaller in terms of fill-in metric than in terms of operation-count metric, as expected. As seen in Table 5.5, for general matrices, oPaToH-MS using $\mathcal{H}^2$,$\mathcal{H}^3$ and $\mathcal{H}^4$ produce orderings with 3%, 6% and 8% less nonzeros in factor matrices than onmetis, respectively, on the average. For LP matrices, oPaToH-MS using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ produces orderings with 9% less nonzeros in factor matrices than onmetis, on the average. Since oPaToH and onmetis are HP-based and GPVS-based ordering tools, respectively, the

TABLE 5.4
Operation counts of various ordering methods and tools relative to onmetis

				oPaToH-MS				oPaToH-ND			
name	MMD	oe	SM	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H_{RN}}(\tilde{A})$	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H_{RN}}(\tilde{A})$
General Matrices											
ncvxqp9	1.42	1.13	1.24	0.74	0.74	0.74	-	0.72	0.72	0.72	-
aug3dcqp	1.63	1.03	1.01	0.92	0.92	0.92	-	0.84	0.84	0.84	-
c-53	1.72	3.11	1.34	0.85	0.85	0.85	-	0.86	0.86	0.86	-
c-59	1.36	1.25	1.34	1.17	1.17	1.17	-	1.07	1.07	1.07	-
c-67	0.85	2.43	1.10	0.88	0.88	0.88	-	0.91	0.91	0.91	-
lshp3025	1.11	0.98	1.05	0.97	1.02	1.02	-	0.99	0.99	0.99	-
lshp3466	1.20	1.09	1.05	0.99	0.91	0.91	-	1.06	0.99	0.99	-
bodyy4	1.55	1.04	1.10	1.04	1.03	1.03	-	1.03	1.02	1.02	-
rail_20209	1.18	1.13	1.21	1.01	0.98	0.98	-	1.10	1.06	1.06	-
cvxbqp1	7.48	0.95	1.08	0.99	0.98	0.98	-	0.91	0.92	0.92	-
shuttle_eddy	1.34	1.00	0.95	1.09	1.06	1.04	-	1.12	1.09	1.11	-
nasa4704	0.74	0.91	0.86	0.63	0.70	0.67	-	0.66	0.68	0.66	-
bcsstk24	0.90	1.11	0.79	0.91	0.83	0.82	-	0.85	0.86	0.87	-
skirt	1.23	1.04	0.92	1.09	1.00	0.97	-	1.20	1.07	1.05	-
bcsstk28	0.65	0.80	0.70	0.76	0.78	0.76	-	0.72	0.77	0.78	-
s1rmq4m1	1.11	0.80	0.84	0.93	0.92	0.91	-	0.89	0.87	0.96	-
vibrobox	0.62	0.67	0.97	0.79	0.56	0.53	-	0.93	0.58	0.52	-
crystk01	1.37	0.82	0.86	1.20	1.12	1.17	-	1.18	1.07	1.10	-
bcsstm36	0.95	0.97	1.14	0.76	0.75	0.77	-	0.79	0.80	0.79	-
gridgena	1.18	0.96	0.99	0.88	0.84	0.84	-	1.01	0.97	0.97	-
k1_san	3.19	1.02	0.77	1.30	0.86	0.93	-	1.36	0.84	0.94	-
finan512	20.71	1.00	1.05	0.95	0.91	0.80	-	0.77	0.82	0.71	-
msc23052	1.01	1.15	1.10	0.84	0.85	0.85	-	0.86	0.90	0.89	-
bcsstk35	0.87	1.22	1.14	0.82	0.80	0.81	-	0.84	0.85	0.84	-
oilpan	1.28	1.00	1.08	0.95	0.97	1.00	-	0.98	0.99	1.04	-
geomean	**1.40**	**1.08**	**1.01**	**0.92**	**0.89**	**0.88**	**-**	**0.93**	**0.89**	**0.89**	**-**
1–,6–,11–				**0.92**	**0.88**	**0.84**	**-**	**0.93**	**0.90**	**0.87**	**-**
LP Problems											
lp_pds_02	1.01	1.03	1.18	0.85	0.84	0.82	0.77	1.41	0.81	0.76	0.81
delf_A_36	0.99	1.18	1.01	0.98	0.86	0.87	0.80	0.96	1.00	0.95	0.85
lp_dfl001	1.49	1.65	4.17	0.75	0.72	0.70	0.72	0.67	0.64	0.63	0.62
model9	0.91	0.85	0.88	0.68	0.67	0.67	0.70	0.69	0.68	0.67	0.72
nl	0.92	30.05	0.96	0.87	0.88	0.90	0.87	0.95	0.93	0.94	0.94
ge	1.64	1.22	1.04	1.15	1.19	0.99	0.93	1.22	1.15	0.97	0.91
lp_ken_13	1.00	18.36	1.09	0.98	0.97	0.97	0.97	0.98	0.97	0.97	0.97
lpi_gosh	1.14	1.03	1.04	0.96	0.88	0.84	0.81	1.13	0.92	0.81	0.81
cq9	1.43	43.63	1.33	1.00	0.99	1.00	0.91	1.05	1.05	1.06	0.98
lp_osa_14	1.06	1.00	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06	1.06
co9	1.32	50.74	0.96	0.96	0.97	0.97	0.96	1.00	0.98	1.01	0.98
pltexpa	8.21	1.82	3.34	0.78	0.68	0.67	0.68	0.85	0.84	0.85	0.83
model10	1.85	1.09	2.13	1.06	0.99	0.97	0.99	0.98	0.94	0.94	0.96
fome12	2.00	1.84	5.00	0.93	0.93	0.91	0.91	0.77	0.72	0.72	0.72
lp_cre_d	1.51	0.83	9.01	0.87	0.90	0.89	0.84	1.00	0.98	0.99	0.94
r05	0.62	12.63	0.60	0.44	0.44	0.44	0.44	0.44	0.44	0.44	0.44
world	0.81	2.04	2.32	0.71	0.71	0.71	0.64	0.96	0.87	0.86	0.77
mod2	0.62	1.76	1.77	0.57	0.55	0.55	0.52	0.77	0.69	0.70	0.66
lp_maros_r7	0.74	0.94	1.10	1.10	1.09	1.02	0.92	1.14	1.11	1.02	0.92
ex3sta1	9.51	1.03	1.26	1.35	1.22	1.59	1.03	1.43	1.24	1.73	0.94
psse2	0.73	1.25	0.91	0.68	0.70	0.65	0.62	0.80	0.80	0.76	0.72
fxm3_16	0.72	1.09	1.23	0.72	0.74	0.72	0.73	0.89	0.93	0.92	0.94
Kemelmacher	1.93	0.87	1.04	1.65	1.28	1.14	0.89	1.78	1.29	1.11	0.91
graphics	1.13	1.02	1.07	0.83	0.82	0.84	0.73	0.89	0.89	0.90	0.77
stat96v5	0.76	1.04	0.76	0.94	0.98	0.88	0.79	0.93	0.97	0.89	0.79
geomean	**1.27**	**2.22**	**1.42**	**0.88**	**0.86**	**0.84**	**0.79**	**0.96**	**0.89**	**0.88**	**0.83**

TABLE 5.5
Factor nonzero counts of various ordering methods and tools relative to onmetis

				oPaToH-MS				oPaToH-ND			
name	MMD	oe	SM	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H}_{\mathrm{RN}}(\tilde{A})$	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H}_{\mathrm{RN}}(\tilde{A})$
General Matrices											
ncvxqp9	1.09	1.05	1.14	0.95	0.95	0.95	-	0.95	0.95	0.95	-
aug3dcqp	1.12	1.01	1.00	0.90	0.90	0.90	-	0.87	0.87	0.87	-
c-53	1.40	1.45	1.15	1.07	1.07	1.07	-	1.07	1.07	1.07	-
c-59	1.08	1.15	1.10	0.97	0.97	0.97	-	0.96	0.96	0.96	-
c-67	1.03	1.22	1.05	1.02	1.02	1.02	-	1.03	1.03	1.03	-
lshp3025	1.04	0.99	1.01	0.99	1.00	1.00	-	0.99	0.99	0.99	-
lshp3466	1.06	1.03	1.01	0.99	0.97	0.97	-	1.01	0.99	0.99	-
bodyy4	1.15	1.01	1.02	1.01	1.00	1.00	-	1.01	1.00	1.00	-
rail_20209	1.07	1.04	1.07	1.02	1.01	1.01	-	1.04	1.03	1.03	-
cvxbqp1	2.12	0.98	1.04	0.97	0.97	0.97	-	0.96	0.97	0.97	-
shuttle_eddy	1.13	1.00	0.97	1.03	1.01	1.01	-	1.03	1.02	1.03	-
nasa4704	0.85	0.96	0.94	0.81	0.83	0.82	-	0.82	0.83	0.82	-
bcsstk24	0.93	1.05	0.91	0.94	0.91	0.91	-	0.92	0.93	0.93	-
skirt	1.06	1.02	0.95	1.02	0.99	0.98	-	1.05	1.01	1.01	-
bcsstk28	0.85	0.96	0.88	0.89	0.90	0.90	-	0.88	0.90	0.90	-
s1rmq4m1	1.03	0.92	0.94	0.96	0.96	0.95	-	0.95	0.95	0.98	-
vibrobox	0.82	0.82	1.01	0.85	0.74	0.72	-	0.91	0.75	0.72	-
crystk01	1.12	0.91	0.91	1.06	1.03	1.05	-	1.06	1.02	1.03	-
bcsstm36	0.96	0.98	1.00	0.90	0.90	0.90	-	0.91	0.91	0.91	-
gridgena	1.09	0.99	0.96	0.96	0.95	0.94	-	1.00	0.98	0.98	-
k1_san	1.66	1.00	0.90	1.12	0.94	0.97	-	1.14	0.94	0.97	-
finan512	2.88	1.02	1.07	1.02	0.99	0.96	-	0.96	0.96	0.92	-
msc23052	0.95	1.03	0.97	0.90	0.90	0.91	-	0.91	0.92	0.91	-
bcsstk35	0.92	1.06	1.00	0.90	0.90	0.90	-	0.91	0.91	0.91	-
oilpan	1.05	1.01	1.02	0.96	0.97	0.98	-	0.97	0.97	0.99	-
geomean	**1.13**	**1.02**	**1.00**	**0.97**	**0.95**	**0.95**	**-**	**0.97**	**0.95**	**0.95**	**-**
				0.97	**0.94**	**0.92**	**-**	**0.97**	**0.95**	**0.93**	**-**
LP Problems											
lp_pds_02	1.01	0.98	1.09	0.97	0.96	0.95	0.93	1.11	0.95	0.93	0.95
delf_A_36	0.99	1.07	1.00	0.98	0.95	0.95	0.93	0.98	0.99	0.97	0.94
lp_dfl001	1.14	1.30	2.10	0.85	0.83	0.82	0.83	0.82	0.80	0.79	0.79
model9	0.98	0.93	0.94	0.86	0.86	0.86	0.87	0.87	0.86	0.86	0.88
nl	0.95	5.18	0.99	0.94	0.94	0.95	0.94	0.96	0.96	0.96	0.96
ge	1.12	1.06	1.03	1.01	1.01	0.97	0.96	1.03	1.00	0.97	0.95
lp_ken_13	1.02	2.17	1.08	1.02	1.02	1.02	1.02	1.03	1.02	1.02	1.02
lpi_gosh	1.01	1.03	1.00	0.97	0.94	0.94	0.92	1.02	0.95	0.93	0.92
cq9	1.12	4.74	1.10	0.99	0.99	0.99	0.96	1.01	1.01	1.01	0.98
lp_osa_14	1.02	1.00	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02	1.02
co9	1.07	5.23	0.96	0.97	0.97	0.97	0.96	0.98	0.97	0.98	0.97
pltexpa	2.04	1.33	1.90	0.85	0.81	0.81	0.81	0.86	0.85	0.85	0.84
model10	1.27	1.05	1.38	1.03	1.00	0.99	0.99	1.00	0.98	0.98	0.99
fome12	1.33	1.39	2.30	0.95	0.95	0.94	0.93	0.88	0.85	0.85	0.85
lp_cre_d	1.17	0.93	2.56	0.93	0.95	0.94	0.92	0.98	0.98	0.98	0.96
r05	0.93	3.33	0.92	0.82	0.82	0.82	0.82	0.82	0.82	0.82	0.82
world	0.90	1.39	1.48	0.86	0.86	0.86	0.83	0.95	0.92	0.91	0.88
mod2	0.82	1.32	1.32	0.79	0.78	0.78	0.76	0.87	0.84	0.84	0.82
lp_maros_r7	0.87	0.97	1.03	1.03	1.03	1.00	0.95	1.05	1.04	1.00	0.96
ex3sta1	3.23	1.00	1.13	1.12	1.09	1.26	0.99	1.14	1.09	1.31	0.95
psse2	0.87	1.07	0.95	0.87	0.87	0.86	0.84	0.90	0.90	0.89	0.87
fxm3_16	0.93	1.03	1.07	0.93	0.94	0.93	0.94	0.97	0.98	0.97	0.98
Kemelmacher	1.31	0.96	1.02	1.23	1.10	1.04	0.94	1.28	1.10	1.03	0.95
graphics	0.98	0.99	0.97	0.89	0.89	0.89	0.85	0.91	0.91	0.92	0.87
stat96v5	0.88	1.02	0.88	0.97	0.99	0.94	0.90	0.97	0.99	0.95	0.90
geomean	**1.10**	**1.40**	**1.19**	**0.95**	**0.94**	**0.94**	**0.91**	**0.97**	**0.95**	**0.95**	**0.92**

better quality orderings produced by oPaToH confirm the validity of our HP-based GPVS formulation in the application of fill reducing ordering of sparse matrices.

As seen in Table 5.6, for general matrices, searching for 3-cliques in the construction of clique-node hypergraphs amortizes its cost in 11 out of 20 matrices by reducing the total ordering time. oPaToH-MS using $\mathcal{H}^3$ takes 3.2% less ordering time than oPaToH-MS using $\mathcal{H}^2$, on the average. However, as seen in the table, searching for 4-cliques in the construction of clique-node hypergraphs amortizes its cost in only 6 out of 15 matrices. oPaToH-MS using $\mathcal{H}^4$ takes 8.8% more ordering time than oPaToH-MS using $\mathcal{H}^2$, on the average. As seen in the table, oPaToH-MS is significantly slower than onmetis for the ordering of general matrices. For example, oPaToH-MS using $\mathcal{H}^3$ is 263% slower than onmetis, on the average. However, for LP matrices, oPaToH-MS using $\mathrm{H}_{\mathrm{RN}}(\tilde{A})$ is quite fast and it is only 79% slower than onmetis, on the average. The slower run-time performance of oPaToH compared to onmetis is expected, because hypergraph partitioning is computationally more expensive than graph partitioning, in general.

The above discussions given on Tables 5.4–5.6 show that oPaToH produces considerably better quality orderings than onmetis at a higher computational cost. Thus, the higher computational cost of oPaToH can be typically justified for applications which involve multiple numerical factorization of matrices with the same sparsity patterns and/or multiple solutions with different right hand side vectors. Interior point methods that adopt the normal equations approach constitute such a typical case. This is because the numerical factorization $M = LL^T$ of matrix M is required at each iteration, where the sparsity pattern of matrix M is independent of the value of diagonal D^2 matrix and hence remains same at all iterations.

6. Conclusion. Direct solvers are one of the preferred methods for solving linear systems due to their numerical robustness. A typical first step in this process is reordering of input matrix to improve execution time and space requirements of the solution process. Graphs have been extensively used to model the evolution of the nonzero structure during the factorization step of direct solvers and hence for the reordering process. Decades after the first theoretical work on nested dissection, recent advances in multilevel graph partitioning framework finally enabled the development of long-waited, successful nested dissection based ordering tools that work in wider range of problems. The state-of-the-art nested dissection based ordering tools directly employ graph partitioning by vertex separator (GPVS). In this work, we showed that GPVS has a deficiency in multilevel frameworks. We introduced a novel hypergraph partitioning (HP) formulation of GPVS that is not vulnerable to GPVS's deficiency in multilevel framework. We have exploited this finding to develop a novel HP-based fill reducing ordering method. In matrix terms, our approaches rely on existence of a structural factorization of a symmetric matrix M in the form of $M = AA^T$, where A is a rectangular matrix. Such structural factorizations arise in different contexts, such as solution of LP problems, where $M = AD^2A^T$ and D^2 is a diagonal matrix. In the absence of such structural factorization, we also proposed simple, yet effective structural factorization techniques that can be applied to any arbitrary symmetric matrix to obtain such structural factorization. For matrices coming from LP problems, we also proposed two structural factor sparsening methods.

We performed our experimental evaluations using 50 publicly available test matrices, where 25 of them come from LP problems, and 25 are general symmetric matrices. We have implemented and tested multisection (MS) and nested-dissection (ND) ordering schemes [7] in our HP-based ordering tool, oPaToH, and compared our results to MS implementation in SMOOTH [5], as well as GPVS- and GPES-based orderings implemented in MeTiS [37], MMD [40] implementation from SPARSPAK [23]. Among the existing tools we tested, in general, GPVS-based onmetis produces best results in terms of operation counts and amount

TABLE 5.6
Total execution times of various ordering methods and tools relative to onmetis

				oPaToH-MS				oPaToH-ND			
name	MMD	oe	SM	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H_{RN}}(\tilde{A})$	$\mathcal{H}^2$	$\mathcal{H}^3$	$\mathcal{H}^4$	$\mathrm{H_{RN}}(\tilde{A})$
General Matrices											
ncvxqp9	0.63	0.69	2.31	2.06	2.06	2.06	-	2.07	2.07	2.07	-
aug3dcqp	1.70	0.80	1.61	1.30	1.30	1.30	-	1.30	1.30	1.30	-
c-53	9.04	0.98	15.51	3.41	3.41	3.41	-	3.16	3.16	3.16	-
c-59	7.68	0.73	8.27	3.77	3.77	3.77	-	3.66	3.66	3.66	-
c-67	1.66	0.91	5.08	9.40	9.40	9.40	-	9.61	9.61	9.61	-
lshp3025	1.00	1.00	4.00	9.30	6.00	6.00	-	10.30	6.00	6.00	-
lshp3466	1.00	1.00	3.00	5.70	3.50	3.50	-	5.65	3.65	3.65	-
bodyy4	0.83	0.83	1.94	4.45	2.62	2.62	-	4.26	2.49	2.49	-
rail_20209	0.63	0.83	1.83	4.18	2.44	2.44	-	4.37	2.45	2.45	-
cvxbqp1	1.48	0.86	2.02	5.52	3.47	3.43	-	5.52	3.56	3.65	-
shuttle_eddy	0.64	0.82	1.91	6.43	4.05	3.53	-	6.34	3.98	3.75	-
nasa4704	0.67	2.00	4.67	4.10	4.57	4.70	-	4.50	4.63	4.73	-
bcsstk24	0.50	3.00	8.50	3.85	5.05	8.30	-	3.60	5.60	8.25	-
skirt	0.65	0.88	2.24	8.40	5.26	4.23	-	8.51	5.21	4.30	-
bcsstk28	0.33	3.33	7.67	1.93	3.23	5.03	-	1.87	3.20	5.03	-
s1rmq4m1	0.67	4.33	7.33	2.30	5.03	7.43	-	2.27	5.10	7.40	-
vibrobox	1.21	0.79	4.91	10.14	6.61	5.45	-	9.98	6.56	5.45	-
crystk01	0.50	2.50	7.17	4.03	5.97	8.05	-	4.23	5.80	8.33	-
bcsstm36	0.18	2.29	4.16	0.85	0.98	1.03	-	0.84	0.92	1.00	-
gridgena	0.74	0.89	1.68	6.63	4.46	3.73	-	6.21	4.15	3.66	-
k1_san	0.44	0.83	1.54	4.61	2.47	1.95	-	4.55	2.49	1.92	-
finan512	0.79	0.78	1.91	5.08	2.98	1.72	-	5.22	2.98	1.61	-
msc23052	0.32	3.27	5.59	1.82	3.52	4.52	-	1.82	3.51	4.47	-
bcsstk35	0.43	3.13	6.67	2.43	3.56	5.57	-	2.46	3.48	5.53	-
oilpan	0.42	4.23	8.52	1.45	2.60	4.69	-	1.49	2.57	4.72	-
geomean	**0.81**	**1.34**	**3.86**	**3.75**	**3.52**	**3.74**	**-**	**3.75**	**3.50**	**3.73**	**-**
1–,6–,11–				**3.75**	**3.63**	**4.08**	**-**	**3.75**	**3.61**	**4.07**	**-**
LP Problems											
lp_pds_02	2.00	1.00	5.00	5.80	4.95	5.05	5.10	5.40	4.90	5.60	4.70
delf_A_36	0.67	0.67	3.00	5.17	3.80	3.57	1.70	5.13	3.97	3.50	1.70
lp_dfl001	6.08	0.67	4.92	5.28	3.97	3.76	2.44	5.23	3.92	3.48	2.50
model9	0.83	1.17	2.33	9.35	8.05	7.22	2.62	9.43	8.17	7.30	2.63
nl	3.67	1.00	9.67	11.19	7.06	6.50	2.30	11.18	7.00	6.28	2.24
ge	0.64	0.71	3.21	4.78	3.41	2.82	1.48	4.56	3.31	2.82	1.49
lp_ken_13	1.80	0.97	2.63	6.10	3.49	4.06	3.42	6.04	3.67	4.14	3.37
lpi_gosh	2.18	0.91	6.64	17.22	13.65	11.83	3.33	17.40	13.55	11.86	3.24
cq9	4.06	1.00	9.29	20.19	13.43	12.53	2.75	20.29	13.07	12.11	2.86
lp_osa_14	1.00	0.93	20.79	0.55	11.86	34.02	0.92	0.54	11.90	34.02	0.89
co9	5.05	1.00	10.60	19.63	13.75	11.43	2.51	19.76	13.17	12.04	2.52
pltexpa	0.90	0.78	1.50	4.26	2.80	2.53	1.80	4.29	2.77	2.47	1.78
model10	1.20	1.20	5.80	15.15	13.97	12.62	4.03	15.23	13.64	11.92	4.00
fome12	4.96	0.70	3.97	7.98	4.89	4.07	2.78	7.96	4.79	3.86	2.71
lp_cre_d	4.06	0.74	7.66	47.90	26.63	19.69	13.35	40.53	28.21	18.94	13.20
r05	0.47	1.12	2.82	6.64	4.59	5.24	1.53	6.56	4.42	5.14	1.58
world	2.96	0.87	3.73	11.95	7.00	5.54	1.77	11.52	7.08	5.24	1.67
mod2	2.87	0.83	4.44	11.18	7.04	5.34	1.60	11.24	6.79	5.25	1.64
lp_maros_r7	0.56	0.84	9.69	51.00	31.61	30.35	1.23	52.42	32.00	30.12	1.28
ex3sta1	1.56	1.09	17.64	8.18	7.81	6.34	0.82	8.23	7.76	6.02	0.80
psse2	0.41	2.64	7.55	2.50	3.34	4.38	0.85	2.40	3.38	4.44	0.83
fxm3_16	0.58	0.87	2.96	7.72	5.78	5.42	1.42	7.69	6.13	5.10	1.42
Kemelmacher	0.35	0.80	2.51	11.74	6.36	5.46	0.91	12.15	6.94	5.57	0.89
graphics	0.38	2.69	6.77	3.04	4.86	6.59	0.71	3.16	4.82	6.59	0.69
stat96v5	4.25	1.00	4.97	175.4	104.9	82.98	0.16	172.7	108.2	83.00	0.16
geomean	**1.46**	**0.98**	**5.19**	9.38	7.89	7.62	**1.79**	9.27	7.92	7.53	**1.77**

of fill-in. In terms of operation counts, our HP-based tool oPaToH-MS produced orderings that require 16% and 21% less operation counts than the ones produced by onmetis for general and LP matrices, respectively, on the average. In terms of number of nonzero counts in the triangular factors, oPaToH-MS produces 8-9% less nonzeros in comparison to onmetis. These reductions come at the expense of higher execution time. oPaToH is up to 4 times slower than onmetis on general symmetric matrices, however it is only 1.8 times slower for ordering LP matrices. These higher computational costs can be easily amortized in applications involving multiple numerical factorization of matrices with the same sparsity patterns and/or multiple solutions with different right hand side vectors.

REFERENCES

[1] C. J. ALPERT AND A. B. KAHNG, *Recent directions in netlist partitioning: A survey*, VLSI Journal, 19 (1995), pp. 1–81.

[2] P. AMESTOY, T. A. DAVIS, AND I. S. DUFF, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[3] E. D. ANDERSEN, J. GONDZIO, C. MESZAROS, AND X. XU, *Implementation of interior point methods for large scale linear programming*, in in Interior Point Methods in Mathematical Programming, Kluwer Academic Publishers, 1996, pp. 189–252.

[4] C. ASHCRAFT AND R. GRIMES, *Spooles: An object-oriented sparse matrix library*, in In Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.

[5] C. ASHCRAFT AND J. W. H. LIU, *SMOOTH: A software package for ordering sparse matrices*, 1996.

[6] ———, *Applications of the dulmage-mendelsohn decomposition and network flow to graph bisection improvement*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 325–354.

[7] ———, *Robust ordering of sparse matrices using multisection*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 816–832.

[8] C. AYKANAT, A. PINAR, AND U. V. ÇATALYÜREK, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM Journal on Scientific Computing, 26 (2004), pp. 1860–1879.

[9] A. BRANDSTADT, V. B. LE, AND J. P. SPINRAD, *Graph Classes: A Survey*, SIAM, 1999.

[10] T. N. BUI AND C. JONES, *A heuristic for reducing fill-in sparse matrix factorization*, in Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing, SIAM, 1993, pp. 445–452.

[11] U. V. CATALYUREK, C. AYKANAT, AND B. UÇAR, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, Tech. Report OSUBMI_TR_2008_n04, The Ohio State University, Department of Biomedical Informatics, 2008. submitted to SIAM J. Sci. Comput.

[12] U. V. ÇATALYÜREK, *Hypergraph Models for Sparse Matrix Partitioning and Reordering*, PhD thesis, Bilkent University, Computer Engineering and Information Science, Nov 1999. Available at `http://www.cs.bilkent.edu.tr/tech-reports/1999/ABSTRACTS.1999.html`.

[13] U. V. ÇATALYÜREK AND C. AYKANAT, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

[14] ———, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at `http://bmi.osu.edu/~umit/software.htm`, 1999.

[15] ———, *A fine-grain hypergraph model for 2D decomposition of sparse matrices*, in Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA, April 2001.

[16] ———, *Hypergraph-partitioning-based sparse matrix ordering*, in Second International Workshop on Combinatorial Scientific Computing (CSC05), CERFACS, Toulouse, France, Jun 2005.

[17] C. CHEVALIER AND F. PELLEGRINI, *PT-SCOTCH: A tool for efficient parallel graph ordering*, Parallel Computing, 34 (2008), pp. 318–331.

[18] J. CONG, W. LABIO, AND N. SHIVAKUMAR, *Multi-way vlsi circuit partitioning based on dual net representation*, in Proceedings of IEEE International Conference on Computer-Aided Design, 1994, pp. 56–62.

[19] T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST, *Introduction to Algorithms*, MIT Press, McGraw-Hill, New York, NY, 1990.

[20] G. B. DANTZIG AND M. N. THAPA, *Linear Programming: Theory and extensions*, Springer, 2003.

[21] T. DAVIS, *University of Florida sparse matrix collection:* `http://www.cise.ufl.edu/research/sparse/`, NA Digest, 92/96/97 (1994/1996/1997).

[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, 1982, pp. 175–181.

[23] A. GEORGE AND E. NG, *A new release of sparspak: the waterloo sparse matrix package*, SIGNUM Newsl.,

19 (1984), pp. 9–13.
[24] J. A. GEORGE, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.
[25] J. A. GEORGE AND J. W. H. LIU, *Computer solution of large sparse positive definite systems*, Prentice-Hall, 1981.
[26] J. GRAMM, J. GUO, F. HÜFFNER, AND R. NIEDERMEIER, *Data reduction, exact, and heuristic algorithms for clique cover*, in Proceedings of the Eighth Workshop on Algorithm Engineering, 2006.
[27] L. GRIGORI, E. BOMAN, S. DONFACK, AND T. DAVIS, *Hypergraph unsymmetric nested dissection ordering for sparse LU factorization*, Tech. Report 2008-1290J, Sandia National Labs, 2008. Submitted to SIAM J. Sci. Comp.
[28] A. GUPTA, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, Tech. Report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.
[29] ———, *Watson graph partitioning package*, Tech. Report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.
[30] B. HENDRICKSON AND T. G. KOLDA, *Graph partitioning models for parallel computing*, Parallel Computing, 26 (2000), pp. 1519–1534.
[31] B. HENDRICKSON AND R. LELAND, *A multilevel algorithm for partitioning graphs*, in Proc. Supercomputing '95, ACM, December 1995.
[32] B. HENDRICKSON AND E. ROTHBERG, *Effective sparse matrix ordering: just around the bend*, in Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing, 1997.
[33] A. B. KAHNG, *Fast hypergraph partition*, in Proceedings of the 26th ACM/IEEE Design Automation Conference, 1989, pp. 762–766.
[34] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, 1972, pp. 85–103.
[35] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.
[36] ———, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, 20 (1998), pp. 359–392.
[37] ———, *MeTiS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.
[38] E. KELLERMAN, *Determination of keyword conflict*, IBM Technical Disclosure Bulletin, (1973).
[39] L. KOU, L. STOCKMEYER, AND C.K.WONG, *Covering edges by cliques with regard to keyword conflicts and intersection graphs*, Communications of the ACM, 21 (1978), pp. 135–139.
[40] J. W. H. LIU, *Modification of the minimum degree algorith by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.
[41] J. W. H. LIU, *The minimum degree ordering with constraints*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 1136–1145.
[42] D. J. ROSE, *Graph Theory and Computing*, Academic Press, 1972, ch. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations, pp. 183–217.
[43] E. ROTHBERG, *Ordering sparse matrices using approximate minimum local fill*, in Second SIAM Conference on Sparse Matrices, 1996.
[44] W. F. TINNEY AND J. W. WALKER, *Direct solution of sparse network equations by optimally ordered triangular factorization*, in Proc. IEEE, vol. 55, 1967, pp. 1801–1809.
[45] B. UÇAR AND C. AYKANAT, *Revisiting hypergraph models for sparse matrix partitioning*, SIAM Review, 49 (2007), pp. 595–603.
[46] M. YANNAKIS, *Computing the minimum fill-in is np-complete*, SIAM J. Algebraic Discrete Methods, 2 (1981), pp. 77–79.