A HYPERGRAPH PARTITIONING MODEL FOR PROFILE MINIMIZATION

SEHER ACER[†], ENVER KAYAASLAN[†‡], AND CEVDET AYKANAT[†]

Abstract. In this paper, the aim is to symmetrically permute the rows and columns of a given sparse symmetric matrix so that the profile of the permuted matrix is minimized. We formulate this permutation problem by first defining the m-way ordered hypergraph partitioning (moHP) problem and then showing the correspondence between profile minimization and moHP problems. For solving the moHP problem, we propose a recursive-bipartitioning-based hypergraph partitioning algorithm, which we refer to as the moHP algorithm. This algorithm achieves a linear part ordering via left-to-right bipartitioning. In this algorithm, we utilize fixed vertices and two novel cut-net manipulation techniques in order to address the minimization objective of the moHP problem. We show the correctness of the moHP algorithm and describe how the existing partitioning tools can be utilized for its implementation. Experimental results on an extensive set of matrices show that the moHP algorithm obtains smaller profile than the state-of-the-art profile reduction algorithms, which then results in considerable improvements in the factorization runtime in a direct solver.

Key words. sparse matrices, matrix ordering, matrix profile, matrix envelope, profile minimization, profile reduction, hypergraph partitioning, recursive bipartitioning

AMS subject classifications. 05C50, 05C85, 65F05, 65F50, 68R10

1. Introduction.

The focus of this work is to minimize the envelope size, i.e., *profile*, of a given $m \times m$ sparse symmetric matrix $A = (a_{ij})$ through symmetric row/column permutation. The envelope of A, $E(A)$, is defined as the set of index pairs in each row that lie between the first nonzero entry and the diagonal. That is,

$$E(A) = \{(i, j) : fc(i) \leq j < i,\ i = 1, 2, \ldots, m\},$$

where $fc(i)$ denotes the column index of the first nonzero entry in row i, i.e., $fc(i) = \min\{j : a_{ij} \neq 0\}$. The size of the envelope of A is referred to as the profile of A, which is denoted by $P(A)$. Note that profile can also be expressed as the sum of row widths in envelope, that is,

$$P(A) = |E(A)| = \sum_{i=1}^{m}(i - fc(i)).$$

Diaz et al. [16] describe a number of graph layout problems which are similar or equivalent to the profile minimization problem and the application areas of these problems.

The profile minimization problem arises in various applications. The greatest attention given to this problem is from the scientific computing domain due to improving the performance of the sparse solvers. Basically, sparse Gaussian elimination benefits from an ordering of the input matrix with small profile in terms of both storage and number of floating-point operations [20, 37]. The computational complexity of envelope methods is proportional to the sum of squares of row widths. Similarly, the computational complexity of frontal methods is proportional to the sum of squares of front sizes, where the sum of the profile and the number of rows gives the sum of front sizes. While envelope methods are now outdated, frontal methods and their extensions such as multifrontal ones are still actively used. Davis et al. [14] list these methods in

[†]Computer Engineering Department, Bilkent University, 06800, Ankara, Turkey (acer@cs.bilkent.edu.tr, enver@cs.bilkent.edu.tr, aykanat@cs.bilkent.edu.tr)

[‡]Currently with Google Switzerland, 8002, Zürich, Switzerland

their recent and extensive survey on sparse direct methods. Besides direct methods, small profile is also shown to be desirable for improving the performance of iterative methods, including incomplete factorization preconditioners [12, 15, 19, 24, 39]. Furthermore, improving cache hit rates in sparse matrix computations can be considered as another application for this problem [9, 41]. In addition to the scientific computing domain, the profile metric and the corresponding minimization problem are found to be useful in applications from other domains such as bioinformatics, model checking, and visualization [6, 7, 26, 28, 33, 34].

The profile minimization problem is NP-hard [32]. Heuristics proposed for solving this problem are plentiful in the literature. In the following, we summarize the most commonly-used profile reduction heuristics and refer the reader to the recent systematic review in [5] for a more complete list. The earliest methods such as RCM [21], GPS [23], Gibbs-King [22], and Sloan [40] exploit the level structure obtained on the standard graph representation of the given matrix. Most of the successor methods use a spectral approach [4], which obtain better results compared to the earlier methods at the expense of higher ordering runtimes. These runtimes are improved by hybrid methods [8, 29, 30, 35], which exploit both graph-based and spectral approaches in a multilevel framework. These algorithms include the one proposed by Hu and Scott [29], which obtains smaller profile values that the preceding algorithms. Reid and Scott [36] show that applying Hager's exchange methods [27] as a post-processing step to the algorithm proposed by Hu and Scott [29] achieves even better results.

The contributions of this paper are as follows. We first define an ordered version of hypergraph partitioning (HP) problem, which we referred to as the m-way ordered hypergraph partitioning (moHP) problem. Then, we formulate the profile minimization problem as an moHP problem. To our knowledge, this work is the first in the literature which formulates the profile minimization using an HP problem. For solving the moHP problem, we propose the moHP algorithm, which is based on the recursive bipartitioning (RB) paradigm. The moHP algorithm achieves a linear part ordering via left-to-right bipartitioning. In order to address the minimization objective of the moHP problem, the moHP algorithm utilizes fixed vertices within the RB framework and two novel cut-net manipulation techniques. We theoretically show that minimizing a cost metric in each RB step corresponds to minimizing the objective of the moHP problem. We also show how existing HP tools can be utilized in the proposed RB-based algorithm.

The rest of the paper is organized as follows. Section 2 provides background information. Section 3 presents the moHP problem and shows its correspondence to the profile minimization problem. Section 4 presents the proposed RB-based algorithm for solving the moHP problem, discusses its correctness, and describes the implementation of the proposed algorithm using existing partitioning tools. Section 5 provides the experimental results in comparison with the state-of-the-art profile reduction algorithms and section 6 concludes.

2. Preliminaries.

A *hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of n vertices $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ and a set of m nets $\mathcal{N} = \{n_1, n_2, \ldots, n_m\}$. In $\mathcal{H}$, each net $n_i \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which is denoted by $Pins(n_i)$. The vertices in $Pins(n_i)$ are also referred to as the *pins* of n_i. Each vertex $v_i \in \mathcal{V}$ is assigned a weight, which is denoted by $w(v_i)$. Each net $n_i \in \mathcal{N}$ is assigned a cost, which is denoted by $c(n_i)$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way *partition* of $\mathcal{H}$ if the parts in Π are nonempty, mutually disjoint and exhaustive. For a given partition Π, a net n_i is said to connect

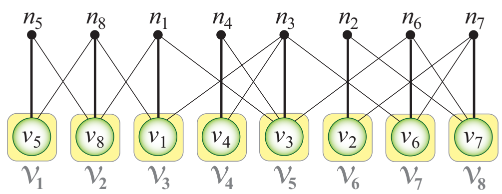

Fig. 1: An m-way ordered partition of a hypergraph $\mathcal{H}$ with $m = 8$ vertices.

a part $\mathcal{V}_k$ if it has pins in $\mathcal{V}_k$, i.e., $Pins(n_i) \cap \mathcal{V}_k \neq \emptyset$. Net n_i is said to be cut if it connects multiple parts in Π, and uncut/internal, otherwise. The *cutsize* of Π is defined as the sum of the costs of the cut nets, that is

$$(1) \qquad cutsize(\Pi) = \sum_{n_i \in \mathcal{N}_c} c(n_i),$$

where $\mathcal{N}_c$ denotes the set of cut nets in Π. The *weight* $W(\mathcal{V}_k)$ of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, i.e., $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$.

Given K and ϵ values, the *hypergraph partitioning (HP)* problem is defined as the problem of finding a K-way partition of a given hypergraph so that the cutsize (1) is minimized and a balance on the weights of the parts is maintained by the constraint

$$(2) \qquad W(\mathcal{V}_k) \leq (1+\epsilon)\frac{\sum_{j=1}^{K} W(\mathcal{V}_j)}{K} \text{ for } k = 1, 2, \ldots, K.$$

Here, ϵ denotes the maximum allowable imbalance ratio on the weights of the parts.

The HP problem with *fixed vertices* is a constrained version of the HP problem where for each part, a subset of vertices can be preassigned to that part before partitioning in such a way that, at the end of the partitioning, they remain in the parts to which they are preassigned. These vertices are called *fixed* vertices. The set of vertices that are fixed to part $\mathcal{V}_k$ is denoted by $\mathcal{F}_k$ for $k = 1, 2, \ldots, K$. The rest of the vertices are called *free* vertices as they can be assigned to any part.

If $K = 2$, then $\Pi = \{\mathcal{V}_1, \mathcal{V}_2\}$ is also referred to as a bipartition. We use $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ to denote a bipartition in which the order of the parts is relevant. Here, $\mathcal{V}_L$ and $\mathcal{V}_R$ respectively denote the left and right parts. In case of bipartitoning with fixed vertices, $\mathcal{F}_L$ and $\mathcal{F}_R$ denote the sets of vertices that are fixed to $\mathcal{V}_L$ and $\mathcal{V}_R$, respectively.

For a given sparse matrix A, the *row-net hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ [10] is formed as follows. As hinted by the name, each row i in A is represented by a net n_i in $\mathcal{N}$. In a dual manner, each column j in A is represented by a vertex v_j in $\mathcal{V}$. For each nonzero entry a_{ij} in A, net n_i connects vertex v_j in $\mathcal{H}$.

3. The m-way ordered hypergraph partitioning formulation. In this section, we first define a variant of the HP problem, the moHP problem, and then show how the profile minimization problem can be formulated as an moHP problem.

3.1. The m-way ordered hypergraph partitioning (moHP) problem. In the moHP problem, we use a special form of partition which is referred to as *m-way ordered partition* (Π_{mo}). Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with m vertices, that is, $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$. A partition of $\mathcal{H}$ is an m-way ordered partition if each part contains exactly one vertex and the parts are subject to an order. We use $\Pi_{mo} = \langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m \rangle$ to denote an m-way ordered partition. Figure 1 displays a sample m-way ordered partition of a hypergraph with $m = 8$ vertices. In this figure, $\mathcal{V}_1 = \{v_5\}$, $\mathcal{V}_2 = \{v_8\}$, and so on. Given an m-way ordered partition Π_{mo}, the *position* of a vertex v_i, $\phi(v_i)$, is defined as the order of the part that contains v_i. That is, $\phi(v_i) = k$ if and only if $\mathcal{V}_k = \{v_i\}$. For example, $\phi(v_1) = 3$ in Figure 1. The *leftmost* vertex f_i of a net n_i is defined as the pin of n_i with the minimum position. That is,

$$f_i = \arg \min_{v_j \in Pins(n_i)} \phi(v_j).$$

For example, $f_3 = v_1$ in Figure 1. The *left span* of a net n_i, $ls(n_i)$, is defined as the difference between the positions of vertices v_i and f_i. That is,

$$ls(n_i) = \phi(v_i) - \phi(f_i). \tag{3}$$

Here, we assume that $v_i \in Pins(n_i)$ for each $n_i \in \mathcal{N}$, thus, $ls(n_i)$ is nonnegative. For example, $ls(n_3) = 5 - 3 = 2$ in Figure 1.

The *cost* of an m-way ordered partition Π_{mo} is defined as the sum of the left spans of the nets in $\mathcal{N}$. That is,

$$cost(\Pi_{mo}) = \sum_{n_i \in \mathcal{N}} ls(n_i). \tag{4}$$

For example, the cost of the m-way partiton in Figure 1 is 8. Note that the cost formulation in (4) is quite different than the traditional cutsize definition in (1).

DEFINITION 1. *(The moHP problem) Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with vertex set $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ and net set $\mathcal{N} = \{n_1, n_2, \ldots, n_m\}$. Assume that $v_i \in Pins(n_i)$ for each net $n_i \in \mathcal{N}$. Then, the moHP problem is the problem of finding an m-way ordered partition Π_{mo} of $\mathcal{H}$ so that the cost given in (4) is minimized.*

3.2. Formulation. The following theorem shows how the profile minimization problem can be formulated as an moHP problem.

THEOREM 2. *Let $\mathcal{H}(A) = (\mathcal{V}, \mathcal{N})$ be the row-net hypergraph of an $m \times m$ structurally symmetric sparse matrix A with $a_{ii} \neq 0$ for $i = 1, 2, \ldots, m$. An m-way ordered partition Π_{om} of $\mathcal{H}(A)$ can be decoded as a row/column permutation P for A so that minimizing the cost of Π_{mo} corresponds to minimizing the profile of the permuted matrix PAP^T.*

Proof. Consider an m-way ordered partition Π_{mo} of $\mathcal{H}(A)$, which is decoded as a row/column permutation for A in such a way that the order of row/column i in the permuted matrix PAP^T is the position $\phi(v_i)$ of vertex v_i in Π_{mo}. That is, the permutation matrix P is formulated as

$$P = \begin{bmatrix} p_1 & p_2 & \cdots & p_m \end{bmatrix},$$

where p_i is a column vector with all zeros except the $\phi(v_i)$'th entry being equal to 1, for $i = 1, 2, \ldots, m$. Consider a row i in PAP^T. Note that a_{ii} is the $\phi(v_i)$'th diagonal entry of PAP^T. Let C_i denote the set of the columns in which row i has a nonzero entry. By the row-net hypergraph formulation, $v_j \in Pins(n_i)$ if and only if $j \in C_i$. Since the order of each column $j \in C_i$ in PAP^T is set to be $\phi(v_j)$, the column representing vertex f_i has the first nonzero entry of row i in PAP^T. Thus, the contribution of row i to the profile of PAP^T is equal to the left span of n_i in Π_{mo}. Hence, the profile of PAP^T is equal to the cost of Π_{mo}. Therefore, minimizing the cost of Π_{mo} corresponds to minimizing the profile of PAP^T. ∎

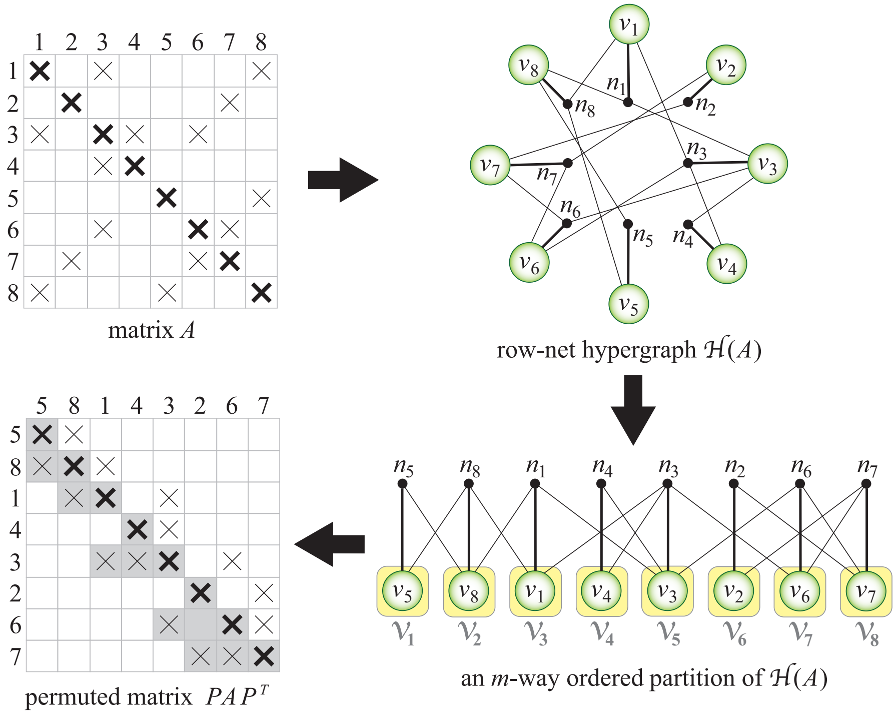

Fig. 2: An illustration for the formulation of the profile minimization problem as an moHP problem.

Figure 2 displays a sample 8×8 structurally symmetric sparse matrix A with 22 nonzero entries and the row-net hypergraph $\mathcal{H}(A)$ of A with 8 vertices, 8 nets and 22 pins. The figure also displays an m-way ordered partition of $\mathcal{H}(A)$ and the permuted matrix PAP^T induced by this partition. For example, consider row 3 in A. As seen in the figure, row 3 is ordered as the fifth row in PAP^T since $\phi(v_3) = 5$. The left span of net n_3, which represents row 3, is computed as $ls(n_3) = \phi(v_3) - \phi(f_3) = 5 - 3 = 2$. Note that the contribution of row 3 to the profile of PAP^T is also 2, which is equal to $ls(n_3)$. The profile of PAP^T is 8, which is equal to the cost of the given m-way ordered partition.

4. Recursive-bipartitioning-based moHP algorithm. This section describes the proposed moHP algorithm, which aims at finding an m-way ordered partition of a given hypergraph with minimum cost. The moHP algorithm is based on the well-known recursive bipartitioning (RB) paradigm and adopts a left-to-right bipartitioning approach. In this approach, a natural order is assumed on the parts of each bipartition and the final partitions of the left and right parts are combined in such a way that their respective orderings are preserved. Recall that the partitioning cost is defined as the sum of the left spans of the nets in (4). Within the left-to-right bipartitioning approach, the moHP algorithm utilizes fixed vertices in order to target the minimization of these left span values.

4.1. Overall description. Algorithm 1 shows the initial invocation of the recursive moHP algorithm. This algorithm first forms the row-net hypergraph $\mathcal{H}(A)$ of the input $m \times m$ structurally symmetric sparse matrix A. In $\mathcal{H}(A)$, each vertex is assigned a unit weight and each net is assigned a unit cost, that is, $w(v_i) = 1$ for each

Algorithm 1 Initial call to the recursive moHP algorithm

Require: $m \times m$ struct. sym. sparse matrix A with nonzero diagonal entries
1: $\mathcal{H}(A) = (\mathcal{V}, \mathcal{N}) \leftarrow$ row-net hypergraph of A
2: $\mathcal{F}_L \leftarrow \mathcal{F}_R \leftarrow \emptyset$
3: $\Pi_{mo} \leftarrow$ moHP$(\mathcal{H}(A), \mathcal{F}_L, \mathcal{F}_R)$ $\quad \triangleright\ \Pi_{mo} = \langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_m \rangle$
4: **for** $i \leftarrow 1$ **to** m **do**
5: $\quad$ Order row/column i as the $\phi(v_i)$'th row/column in PAP^T
6: **return** PAP^T

Algorithm 2 moHP$(\mathcal{H}, \mathcal{F}_L, \mathcal{F}_R)$

Require: Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, fixed-vertex sets $\mathcal{F}_L$ and $\mathcal{F}_R$
1: **if** $\mathcal{V}$ contains exactly one free vertex, say v_i **then**
2: $\quad \Pi_{mo} \leftarrow \langle v_i \rangle$
3: **else**
4: $\quad \Pi \leftarrow$ *bipartition*$(\mathcal{H}, \mathcal{F}_L, \mathcal{F}_R)$ $\quad \triangleright\ \Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$
5: $\quad (\mathcal{H}_L, \mathcal{H}_R, \mathcal{F}_{LR}, \mathcal{F}_{RL}) \leftarrow$ FORM$(\mathcal{H}, \Pi)$
6: $\quad \Pi^L_{mo} \leftarrow$ moHP$(\mathcal{H}_L, \mathcal{F}_L, \mathcal{F}_{LR})$ $\quad \triangleright$ recursive invocation on $\mathcal{H}_L$
7: $\quad \Pi^R_{mo} \leftarrow$ moHP$(\mathcal{H}_R, \mathcal{F}_{RL}, \mathcal{F}_R)$ $\quad \triangleright$ recursive invocation on $\mathcal{H}_R$
8: $\quad \Pi_{mo} \leftarrow \langle \Pi^L_{mo}, \Pi^R_{mo} \rangle$
9: **return** Π_{mo}

$v_i \in \mathcal{V}$ and $c(n_i) = 1$ for each $n_i \in \mathcal{N}$. Then, the moHP algorithm is invoked on $\mathcal{H}(A)$ with empty fixed-vertex sets F_L and $\mathcal{F}_R$, and at the end of this invocation, an m-way ordered partition Π_{mo} of $\mathcal{H}(A)$ is returned. Π_{mo} is then utilized to symmetrically permute the rows and columns of A in such a way that row/colum i is ordered as the $\phi(v_i)$'th row/column in the permuted matrix PAP^T.

Algorithm 2 shows the basic steps of the recursive moHP algorithm. This algorithm takes a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ and fixed-vertex sets $\mathcal{F}_L \subseteq \mathcal{V}$ and $\mathcal{F}_R \subseteq \mathcal{V}$ as input and returns an m'-way ordered partition of $\mathcal{H}$, where m' denotes the number of free vertices in $\mathcal{H}$. Note that $m' = m$ for the initial invocation of this algorithm. The base case and the recursive step of the moHP algorithm are covered in lines 1-2 and 3-8, respectively. In the base case, i.e., when there is exactly one free vertex in $\mathcal{V}$, the singleton partition $\langle v_i \rangle$ is returned, where v_i denotes that free vertex. In the recursive step, i.e., when there are multiple free vertices in $\mathcal{V}$, an ordered bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ of $\mathcal{H}$ is first obtained. In this bipartitioning, the objective is to minimize the left-cut-net metric (5), which is to be explained in section 4.2. The ϵ value to be used in this bipartitioning (see (2)) is investigated in section 5. After Π is obtained, the FORM algorithm is invoked in order to form new hypergraphs $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$ as well as new fixed-vertex sets $\mathcal{F}_{LR}$ and $\mathcal{F}_{RL}$. The details of the FORM algorithm are given in section 4.3. Then, the moHP algorithm is recursively invoked on hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$ to respectively obtain m'_L-way ordered partition Π^L_{mo} of $\mathcal{H}_L$ and an m'_R-way ordered partition Π^R_{mo} of $\mathcal{H}_R$, where m'_L and m'_R respectively denote the numbers of free vertices in $\mathcal{H}_L$ and $\mathcal{H}_R$. Here, $m' = m'_L + m'_R$. Finally, by concatenating Π^L_{mo} and Π^R_{mo}, an m'-way ordered partition Π_{mo} of $\mathcal{H}$ is obtained and returned.

As seen in the recursive invocations of the moHP algorithm in lines 6 and 7, the old fixed-vertex sets $\mathcal{F}_L$ and $\mathcal{F}_R$ associated with the current hypergraph $\mathcal{H}$ are

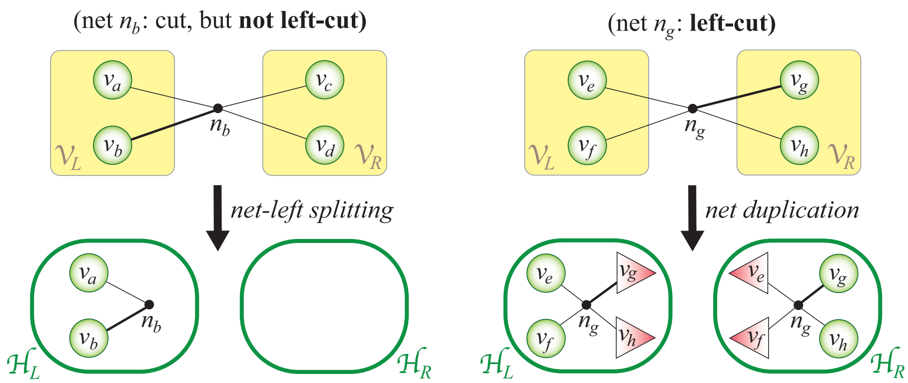

Fig. 3: Upper part: cut nets n_b and n_g. Net n_b is not left-cut since $v_b \in \mathcal{V}_L$, whereas net n_g is left-cut since $v_g \in \mathcal{V}_R$. Lower part: net-left splitting and net duplication are applied on n_b and n_g, respectively.

inherited to the new hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$. That is, the left-fixed-vertex set $\mathcal{F}_L$ and the right-fixed-vertex set $\mathcal{F}_R$ of $\mathcal{H}$ become the left-fixed-vertex set of $\mathcal{H}_L$ and the right-fixed-vertex set of $\mathcal{H}_R$, respectively. In other words, the vertices that become fixed to the left/right part in an invocation of the moHP algorithm remain fixed to the left/right part in the further recursive invocations.

4.2. Left-cut-net metric. Consider the ordered bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ obtained in line 4 of Algorithm 2. Recall that a cut net is defined as a net connecting multiple parts. For encoding the minimization objective of the moHP problem in individual bipartitioning steps, we introduce a special type of cut net, which is referred to as *left-cut net*. A net n_i is said to be a left-cut net if v_i is assigned to $\mathcal{V}_R$ and at least one pin of n_i is assigned to $\mathcal{V}_L$. Figure 3 displays sample cut nets, n_b and n_g, where n_g is a left-cut net while n_b is not.

The set of the left-cut nets, which is denoted by $\mathcal{N}_{\ell c}$, is formulated as

$$\mathcal{N}_{\ell c} = \{n_i : Pins(n_i) \cap \mathcal{V}_L \neq \emptyset \text{ and } v_i \in Pins(n_i) \cap \mathcal{V}_R\}.$$

While obtaining the ordered bipartition Π of $\mathcal{H}$, the objective is to minimize the *left-cut-net* metric, which is defined as the number of left-cut nets in Π, i.e.,

$$(5) \qquad left\text{-}cut\text{-}net(\Pi) = |\mathcal{N}_{\ell c}|.$$

Section 4.4 shows the correctness of this bipartitioning objective in terms of minimizing the cost of the m-way ordered partition obtained by the moHP algorithm, whereas section 4.5 describes how existing partitioning tools can be utilized for encapsulating this bipartitioning objective.

4.3. Forming $\mathcal{H}_L$ and $\mathcal{H}_R$ by novel cut-net manipulation techniques. Algorithm 3 displays the basic steps of the FORM algorithm. As input, it takes a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ and an ordered bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ of $\mathcal{H}$, and it returns new hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$ with fixed-vertex sets $\mathcal{F}_{LR}$ and $\mathcal{F}_{RL}$. This algorithm goes over each net n_i in $\mathcal{N}$ and depending on the distribution of the pins of n_i in Π, it includes n_i in either net set $\mathcal{N}_L$ or net set $\mathcal{N}_R$ or both. If n_i is internal to $\mathcal{V}_L$ (i.e., $Pins(n_i) \subseteq \mathcal{V}_L$), then it is included in $\mathcal{N}_L$ as is. Similarly, if n_i is internal to $\mathcal{V}_R$ (i.e., $Pins(n_i) \subseteq \mathcal{V}_R$), then it is included in $\mathcal{N}_R$ as is. The moHP algorithm handles the

Algorithm 3 FORM($\mathcal{H}, \Pi$)

Require: Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, ordered bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$

1: $\mathcal{N}_L \leftarrow \mathcal{N}_R \leftarrow \emptyset$
2: $\mathcal{F}_{LR} \leftarrow \mathcal{F}_{RL} \leftarrow \emptyset$
3: **for each** $n_i \in \mathcal{N}$ **do**
4:     **if** $Pins(n_i) \subseteq \mathcal{V}_L$ **then** $\triangleright$ n_i is an internal net in $\mathcal{V}_L$
5:         $\mathcal{N}_L \leftarrow \mathcal{N}_L \cup \{n_i\}$
6:     **else if** $Pins(n_i) \subseteq \mathcal{V}_R$ **then** $\triangleright$ n_i is an internal net in $\mathcal{V}_R$
7:         $\mathcal{N}_R \leftarrow \mathcal{N}_R \cup \{n_i\}$
8:     **else if** $v_i \in \mathcal{V}_L$ **then** $\triangleright$ n_i is cut, but not left-cut: *net-left splitting*
9:         $Pins(n_i) \leftarrow Pins(n_i) \cap \mathcal{V}_L$
10:         $\mathcal{N}_L \leftarrow \mathcal{N}_L \cup \{n_i\}$
11:     **else** $\triangleright$ n_i is left-cut: *net duplication*
12:         $leftpins \leftarrow Pins(n_i) \cap \mathcal{V}_L$
13:         $rightpins \leftarrow Pins(n_i) \cap \mathcal{V}_R$
14:         $\mathcal{N}_L \leftarrow \mathcal{N}_L \cup \{n_i\}$
15:         $\mathcal{V}_L \leftarrow \mathcal{V}_L \cup rightpins$
16:         $\mathcal{F}_{LR} \leftarrow \mathcal{F}_{LR} \cup rightpins$ $\triangleright$ *rightpins* are copied to $\mathcal{H}_L$ as right-fixed
17:         $\mathcal{N}_R \leftarrow \mathcal{N}_R \cup \{n_i\}$
18:         $\mathcal{V}_R \leftarrow \mathcal{V}_R \cup leftpins$
19:         $\mathcal{F}_{RL} \leftarrow \mathcal{F}_{RL} \cup leftpins$ $\triangleright$ *leftpins* are copied to $\mathcal{H}_R$ as left-fixed
20: $\mathcal{H}_L \leftarrow (\mathcal{V}_L, \mathcal{N}_L)$
21: $\mathcal{H}_R \leftarrow (\mathcal{V}_R, \mathcal{N}_R)$
22: **return** $\mathcal{H}_L, \mathcal{H}_R, \mathcal{F}_{LR}, \mathcal{F}_{RL}$

cut nets by two novel techniques as follows. If n_i is a cut net, but not a left-cut one (i.e., $v_i \in Pins(n_i) \cap \mathcal{V}_L$ and $Pins(n_i) \cap \mathcal{V}_R \neq \emptyset$), then the *net-left-splitting* technique is applied. In this technique, even though n_i has pins in both $\mathcal{V}_L$ and $\mathcal{V}_R$, it is only included in $\mathcal{N}_L$ with its pins that are assigned to $\mathcal{V}_L$. If n_i is a left-cut net (i.e., $Pins(n_i) \cap \mathcal{V}_L \neq \emptyset$ and $v_i \in Pins(n_i) \cap \mathcal{V}_R$), then the *net-duplication* technique is applied. In this technique, n_i is copied to both $\mathcal{N}_L$ and $\mathcal{N}_R$ with its complete pin set despite the fact that neither $\mathcal{V}_L$ nor $\mathcal{V}_R$ genuinely contains all of n_i's pins. In lines 12 and 13 of the algorithm, *leftpins* and *rightpins* denote the sets of the pins of n_i in $\mathcal{V}_L$ and $\mathcal{V}_R$, respectively. The vertices in *rightpins* are added to vertex set $\mathcal{V}_L$ and they are fixed to the right part of $\mathcal{H}_L$, i.e., included in $\mathcal{F}_{LR}$. In a dual manner, the vertices in *leftpins* are added to vertex set $\mathcal{V}_R$ and they are fixed to the left part of $\mathcal{H}_R$, i.e., included in $\mathcal{F}_{RL}$. After all nets in $\mathcal{N}$ are considered, new hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$ are formed by $\mathcal{H}_L = (\mathcal{V}_L, \mathcal{N}_L)$ and $\mathcal{H}_R = (\mathcal{V}_R, \mathcal{N}_R)$, respectively. As in $\mathcal{H}$, each net in $\mathcal{H}_L$ and $\mathcal{H}_R$ is assigned a unit cost. Each free vertex in $\mathcal{H}_L$ and $\mathcal{H}_R$ is assigned a unit weight, whereas each fixed vertex is assigned a zero weight. Finally, hypergraphs $\mathcal{H}_L$ and $\mathcal{H}_R$ and fixed-vertex sets $\mathcal{F}_{LR}$ and $\mathcal{F}_{RL}$ are returned.

Figure 3 illustrates an example for each of the net-left splitting and net duplication techniques. In the figures throughout the paper, fixed vertices are denoted by triangles pointing a direction, whereas free vertices are denoted by circles. Each vertex fixed to the left part is denoted by a triangle pointing left, whereas each vertex fixed to the right part is denoted by a triangle pointing right. Note that for any net n_i, vertex v_i is *special* compared to the other pins of n_i since the part assignment of v_i determines whether cut net n_i is left-cut or not. Therefore, the connection of n_i to v_i is drawn thicker in the figures for any net n_i.

Figure 4 displays an example for the moHP algorithm run on the hypergraph given in Figure 2. In Figure 4, each rectangular shape with a green border and a white background denotes a hypergraph to be bipartitioned during the moHP algorithm, whereas each rectangular shape with a yellow background denotes a part in an ordered bipartition. To be able to refer the individual hypergraphs, we label them with a Matlab-like notation according to their coverage on the parts of the resulting m-way ordered partition. For example, the initial hypergraph $\mathcal{H}$ is labeled with $\mathcal{H}_{1:8}$ since it covers all eight parts in the resulting m-way ordered partition, while the left and right hypergraphs obtained by bipartitioning $\mathcal{H}_{1:8}$ are labeled with $\mathcal{H}_{1:4}$ and $\mathcal{H}_{5:8}$, respectively. Each left-cut net in the figure is shown in a gray background. Consider the ordered bipartition Π of $\mathcal{H}_{1:8}$. Note that nets n_1, n_3, and n_4 are cut in Π, whereas only n_3 is left-cut among them. Then, $\mathit{left\text{-}cut\text{-}net}(\Pi) = 1$ for this bipartition. Since n_1 and n_4 are cut but not left-cut, the net-left splitting technique is applied on them, that is, they are only included in the left hypergraph $\mathcal{H}_L = \mathcal{H}_{1:4}$ with their respective pins assigned to the left part $\mathcal{V}_L$. Since n_3 is left-cut, the net duplication technique is applied on it, that is, n_3 is included in both hypergraphs $\mathcal{H}_L = \mathcal{H}_{1:4}$ and $\mathcal{H}_R = \mathcal{H}_{5:8}$. Due to the net duplication, the vertices in $rightpins = \{v_3, v_6\}$ are added to the left hypergraph $\mathcal{H}_{1:4}$ as right-fixed, whereas the vertices in $leftpins = \{v_4, v_1\}$ are added to the right hypergraph $\mathcal{H}_{5:8}$ as left-fixed.

4.4. Correctness of the moHP algorithm. In this section, Theorem 7 shows that minimizing the left-cut-net metric (5) in each bipartition of the moHP algorithm corresponds to minimizing the cost (4) of resulting m-way ordered partition. Before that, we provide a brief discussion on the special pins and give some definitions and lemmas to be used in Theorem 7.

We first show that $v_i \in Pins(n_i)$ for each net n_i during the entire moHP algorithm. Note that $v_i \in Pins(n_i)$ for each net in the initial row-net hypergraph. Consider a net n_i in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ on which the moHP algorithm is invoked and assume that $v_i \in Pins(n_i)$ for each $n_i \in \mathcal{N}$. If n_i is included in $\mathcal{H}_L$ or $\mathcal{H}_R$ as is (lines 5 and 7 in Algorithm 3), then $v_i \in Pins(n_i)$ trivially. If net-left splitting is applied on n_i (lines 9-10 in Algorithm 3), then $v_i \in Pins(n_i)$ since $v_i \in \mathcal{V}_L$. If net duplication is applied on n_i (lines 12-19 in Algorithm 3), then $v_i \in Pins(n_i)$ for both copies of n_i in $\mathcal{H}_L$ and $\mathcal{H}_R$ since the whole pin set of n_i is duplicated to $\mathcal{H}_L$ and $\mathcal{H}_R$.

For the nets in the moHP algorithm, we introduce four different states that indicate the connections of the nets to fixed vertices. We call a net n_i

(i) *free*, if it connects no fixed vertices,
(ii) *left-anchored*, if it connects some left-fixed vertices but no right-fixed ones,
(iii) *right-anchored*, if it connects some right-fixed vertices but no left-fixed ones,
(iv) *left-right-anchored*, if it connects some left-fixed and some right-fixed vertices.

Recall that new fixed vertices are only introduced by the net duplication operation and fixed vertices remain fixed in the descendant invocations of the moHP algorithm. Hence, if a net n_i is right-anchored or left-right-anchored, it implies that n_i became left-cut in a bipartition performed in an earlier invocation, and among the two copies of n_i formed after that bipartition, this copy is the one added to the left hypergraph connecting right-fixed vertices that include v_i. Therefore, for each right-anchored or left-right-anchored net n_i, the special pin of n_i, i.e., v_i, is among its right-fixed pins. With a dual reasoning, for each free or left-anchored net n_i, the special pin of n_i is

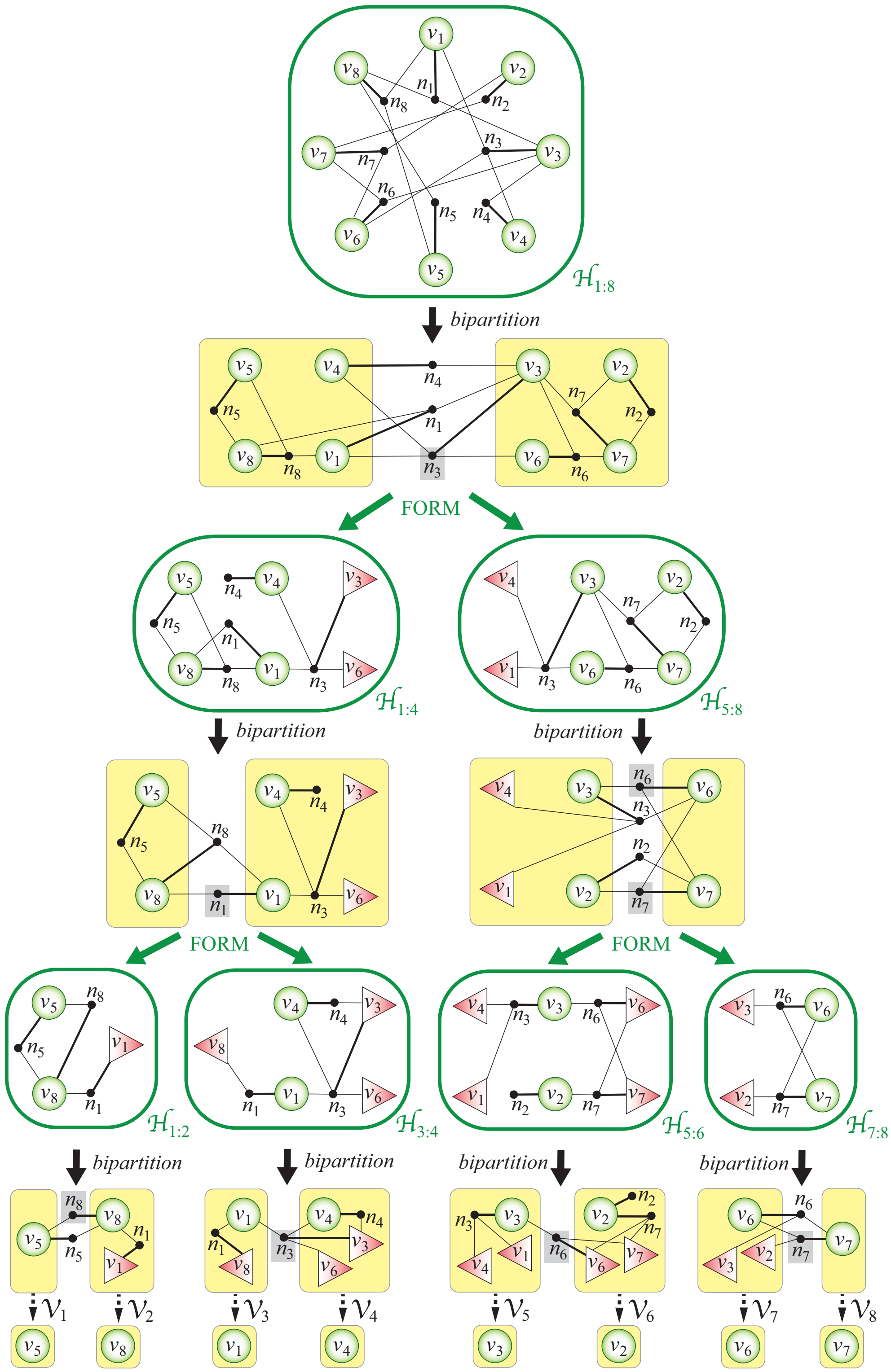

Fig. 4: An example run of the moHP algorithm on the hypergraph given in Figure 2. Left-cut nets are shown in gray background.

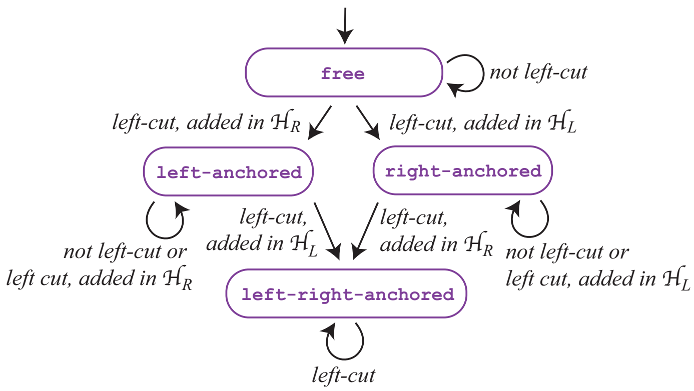

Fig. 5: The state diagram for the states of net n_i in the moHP algorithm.

among its free pins. Finally, for each free or right-anchored net n_i, pin f_i is among its free pins.

Figure 5 displays a state diagram for the states of a net n_i changing through the recursive invocations of the moHP algorithm. Note that all nets are free in the initial invocation of the moHP algorithm, so is n_i. Since the pins of n_i become fixed vertices only after applying net duplication on n_i, n_i stays free as long as it does not become left-cut. If n_i becomes left-cut, net duplication copies it to $\mathcal{H}_L$ and $\mathcal{H}_R$ so that it becomes right-anchored and left-anchored in $\mathcal{H}_L$ and $\mathcal{H}_R$, respectively. Similar to the free nets, left-anchored and right-anchored nets do not change their states until they become left-cut. If a left-anchored net n_i becomes left-cut, then, it becomes left-right-anchored in $\mathcal{H}_L$ while remaining left-anchored in $\mathcal{H}_R$ after net duplication. In a dual manner, if a right-anchored net n_i becomes left-cut, then, it becomes left-right-anchored in $\mathcal{H}_R$ while remaining right-anchored in $\mathcal{H}_L$ after net duplication. Left-right-anchored nets are doomed to become left-cut in all further bipartitionings, hence, a left-right-anchored net n_i remains in the same state in both $\mathcal{H}_L$ and $\mathcal{H}_R$.

The recursive invocations of the moHP algorithm forms a hypothetical full binary tree, which is referred to as an *RB tree* [2, 3, 38]. Each node in the RB tree represents a hypergraph $\mathcal{H}$ on which the moHP algorithm is invoked. If $\mathcal{H}$ contains a single free vertex, which is the base case of the moHP algorithm, then the corresponding node is a leaf node, otherwise, it has one left and one right child nodes, respectively representing $\mathcal{H}_L$ and $\mathcal{H}_R$ obtained in line 5 of Algorithm 2. The RB tree rooted at the node corresponding to hypergraph $\mathcal{H}$ is denoted by $\mathcal{T}^{\mathcal{H}}$. Figure 4 displays a sample RB tree with $m = 8$ leaf nodes.

Given a net n_i in a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ and an RB tree $\mathcal{T}^{\mathcal{H}}$, let $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ denote the number of bipartitions in $\mathcal{T}^{\mathcal{H}}$ in which n_i is left-cut. In the following lemmas and theorem, we abuse the notation and use $\Pi \in \mathcal{T}^{\mathcal{H}}$ to refer to the fact that bipartition Π is performed in one of the nodes of $\mathcal{T}^{\mathcal{H}}$. The following lemmas provide the formulation of $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ for each different state of n_i. Each of Lemmas 3, 4, and 5 is used in the proof(s) of the subsequent lemma(s), whereas Lemma 6 is used in the proof of Theorem. Although we skip the proofs of these lemmas and refer the reader to the Appendix for them, we present all of the lemmas in this section for the sake of completeness. In these lemmas, $\hat{\mathcal{V}}$ denotes the set of free nodes in $\mathcal{H}$, i.e., $\hat{\mathcal{V}} = \mathcal{V} - (\mathcal{F}_L \cup F_R)$.

Lemma 3. *If n_i is left-right-anchored in $\mathcal{H}$, then $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ is equal to the number* *of free nodes in $\mathcal{H}$ minus one, that is,*

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\hat{\mathcal{V}}| - 1.$$

LEMMA 4. *If n_i is right-anchored in $\mathcal{H}$, then $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ is equal to the number of free nodes in $\mathcal{H}$ that are ordered after f_i in Π_{mo}, that is,*

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}|.$$

LEMMA 5. *If n_i is left-anchored in $\mathcal{H}$, then $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ is equal to the number of free nodes at $\mathcal{H}$ that are ordered before v_i in Π_{mo}, that is,*

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) < \phi(v_i)\}|.$$

LEMMA 6. *If n_i is free in $\mathcal{H}$, then $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ is equal to the number of free nodes in $\mathcal{H}$ that are ordered between f_i and v_i in Π_{mo} inclusive minus one, that is,*

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1.$$

THEOREM 7. *Consider a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ on which the moHP algorithm is initially invoked, where $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$, $\mathcal{N} = \{n_1, n_2, \ldots, n_m\}$, and $v_i \in Pins(n_i)$ for each net $n_i \in \mathcal{N}$. Minimizing the left-cut-net metric in each bipartition performed in the moHP algorithm corresponds to minimizing the cost of the resulting m-way ordered partition Π_{mo} of $\mathcal{H}$.*

Proof. Consider an m-way ordered partition Π_{mo} of $\mathcal{H}$ obtained by the moHP algorithm and the left span of a net n_i in $\mathcal{H}$. Note that all nets in $\mathcal{H}$ are free, so is n_i. Recall that $ls(n_i)$ is defined as $\phi(v_i) - \phi(f_i)$ in (3), then,

$$ls(n_i) = \phi(v_i) - \phi(f_i) = |\{v \in \mathcal{V} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1.$$

Then, by Lemma 6,

$$ls(n_i) = \mu(n_i, \mathcal{T}^{\mathcal{H}}), \tag{6}$$

Recall that in (4), $cost(\Pi_{mo})$ is defined as the sum of the left spans of the nets in $\mathcal{H}$, then by (6),

$$cost(\Pi_{mo}) = \sum_{n_i \in \mathcal{N}} ls(n_i) = \sum_{n_i \in \mathcal{N}} \mu(n_i, \mathcal{T}^{\mathcal{H}}).$$

Since $\mu(n_i, \mathcal{T}^{\mathcal{H}})$ is equal to the number of bipartitions in $\mathcal{T}^{\mathcal{H}}$ in which n_i is left-cut, it can also be expressed as

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = \sum_{\Pi \in \mathcal{T}^{\mathcal{H}} : n_i \in \mathcal{N}_{\ell c}^{\Pi}} 1.$$

Here, $\mathcal{N}_{lc}^{\Pi}$ denotes the set of left-cut nets in Π. Then, $cost(\Pi_{mo})$ can be formulated as

$$\begin{aligned} cost(\Pi_{mo}) &= \sum_{n_i \in \mathcal{N}} \mu(n_i, \mathcal{T}^{\mathcal{H}}) = \sum_{n_i \in \mathcal{N}} \sum_{\Pi \in \mathcal{T}^{\mathcal{H}} : n_i \in \mathcal{N}_{\ell c}^{\Pi}} 1 = \sum_{\Pi \in \mathcal{T}^{\mathcal{H}}} \sum_{n_i \in \mathcal{N}_{\ell c}^{\Pi}} 1 \\ &= \sum_{\Pi \in \mathcal{T}^{\mathcal{H}}} left\text{-}cut\text{-}net(\Pi). \end{aligned}$$

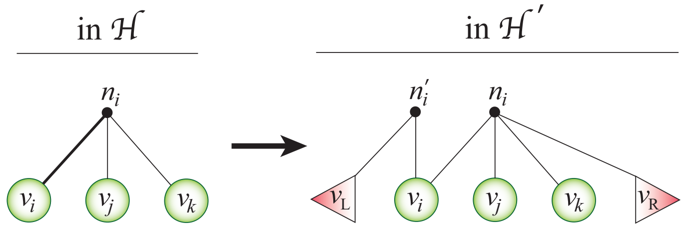

Fig. 6: Net n_i in $\mathcal{H}$ and the net pair (n_i', n_i) added to $\mathcal{H}'$ for n_i.

Since $cost(\Pi_{mo}) = \sum_{\Pi \in \mathcal{T}^{\mathcal{H}}} left\text{-}cut\text{-}net(\Pi)$, minimizing the left-cut-net metric in each bipartition in the moHP algorithm corresponds to minimizing the cost of the resulting m-way partition. ∎

4.5. Minimizing left-cut-net metric. Currently, no existing tool is able to bipartition a given hypergraph with the objective of minimizing the left-cut metric (5). For this reason, in this section, we formulate the bipartitioning problem with the objective of minimizing the left-cut-net metric as an ordinary hypergraph bipartitioning problem with the objective of minimizing the usual cutsize (1).

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a hypergraph which is bipartitioned in line 4 of Algorithm 2. We first transform $\mathcal{H}$ into an extended hypergraph which is denoted by $\mathcal{H}' = (\mathcal{V}', \mathcal{N}')$. In this transformation, we introduce new vertices v_L and v_R to the extended vertex set $\mathcal{V}'$ in addition to the existing ones in $\mathcal{V}$. That is, $\mathcal{V}' = \mathcal{V} \cup \{v_L, v_R\}$. Vertices v_L and v_R are respectively fixed to the left and right parts, so, the fixed-vertex sets $\mathcal{F}_L'$ and $\mathcal{F}_R'$ of $\mathcal{H}'$ are obtained from the fixed-vertex sets $\mathcal{F}_L$ and $\mathcal{F}_R$ of $\mathcal{H}$ by $\mathcal{F}_L' = \mathcal{F}_L \cup \{v_L\}$ and $\mathcal{F}_R' = \mathcal{F}_R \cup \{v_R\}$, respectively. Moreover, for each net $n_i \in \mathcal{N}$, we add an updated version of n_i and a new net n_i' to the extended net set $\mathcal{N}'$. Net n_i is updated by the addition of v_R to its pin set, that is, $Pins(n_i) \leftarrow Pins(n_i) \cup \{v_R\}$. The new net n_i' connects only v_i and v_L, that is, $Pins(n_i') = \{v_i, v_L\}$. Figure 6 displays an example net n_i in $\mathcal{H}$ and the net pair (n_i, n_i') added to $\mathcal{H}'$ for n_i.

A bipartition $\Pi' = \langle \mathcal{V}_L', \mathcal{V}_R' \rangle$ of the extended hypergraph $\mathcal{H}'$ can be decoded as a bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ of $\mathcal{H}$ by simply removing the newly added vertices v_L and v_R from Π'. Note that $v_L \in \mathcal{V}_L'$ and $v_R \in \mathcal{V}_R'$ due to being fixed to the respective part, hence, $\mathcal{V}_L = \mathcal{V}_L' - \{v_L\}$ and $\mathcal{V}_R = \mathcal{V}_R' - \{v_R\}$. The following theorem shows the correspondence between the cutsize (1) of the bipartition Π' of the extended hypergraph $\mathcal{H}'$ and the left-cut-net metric (5) of the ordered bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ of $\mathcal{H}$.

THEOREM 8. *Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ be a hypergraph which is bipartitioned in line 4 of Algorithm 2 and let $\mathcal{H}' = (\mathcal{V}', \mathcal{N}')$ be the corresponding extended hypergraph. Consider a bipartition $\Pi' = \langle \mathcal{V}_L', \mathcal{V}_R' \rangle$ of $\mathcal{H}'$ and the bipartition $\Pi = \langle \mathcal{V}_L, \mathcal{V}_R \rangle$ of $\mathcal{H}$ induced by Π'. Then, minimizing the cutsize of the bipartition Π' (1) corresponds to minimizing the left-cut-net metric in Π (5).*

Proof. We first show the following:

- both n_i and n_i' are cut in Π' if n_i is left-cut in Π (Case 1),
- one of n_i and n_i' is cut in Π', otherwise (Case 2).

(Case 1). Assume that n_i is left-cut in Π. Then, $v_i \in V_R$ and there exists $v_j \in Pins(n_i)$ such that $v_j \in \mathcal{V}_L$. It is clear that $j \neq i$. Then, $v_i \in V_R'$ and $v_j \in V_L'$ in Π'. Thus, n_i' is cut in Π', since it connects both $\mathcal{V}_L'$ and $\mathcal{V}_R'$, respectively due to pins $v_L \in \mathcal{V}_L'$ and $v_i \in \mathcal{V}_R'$. Similarly, n_i is cut in Π' since it connects both $\mathcal{V}_L'$ and $\mathcal{V}_R'$, respectively due to pins $v_j \in \mathcal{V}_L'$ and $v_i \in \mathcal{V}_R'$.

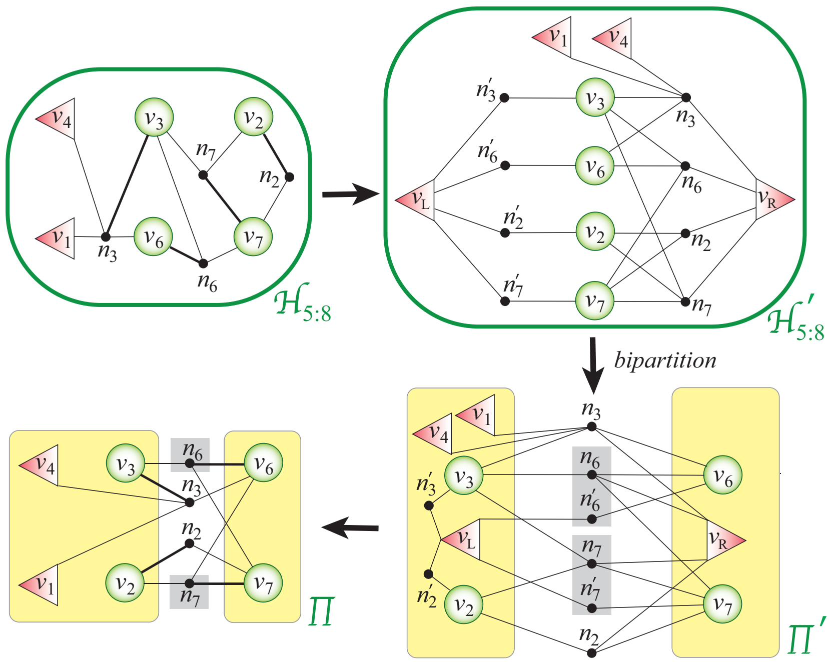

Fig. 7: $\mathcal{H}_{5:8}$ in Figure 4, its extended hypergraph $\mathcal{H}'_{5:8}$, bipartition Π' of $\mathcal{H}'_{5:8}$, and the bipartition Π of $\mathcal{H}_{5:8}$ induced by Π'.

(Case 2.a). Next, assume that n_i is not left-cut in Π and $v_i \in \mathcal{V}_L$. Then, $v_i \in \mathcal{V}'_L$ in Π'. Thus, n'_i is not cut in Π', since it connects only $\mathcal{V}'_L$, i.e., both of its pins (v_i and v_L) reside in $\mathcal{V}'_L$. On the other hand, n_i is cut in Π', since it connects both $\mathcal{V}'_L$ and $\mathcal{V}'_R$, respectively due to pins $v_i \in \mathcal{V}'_L$ and $v_R \in \mathcal{V}'_R$.

(Case 2.b). Finally, assume that n_i is not left-cut in Π and $v_i \in \mathcal{V}_R$. If there existed any pins of n_i in $\mathcal{V}_L$, then n_i would be left-cut, hence, all pins of n_i reside in $\mathcal{V}_R$ in Π. Note that v_R is added to the pin set of n_i in $\mathcal{H}'$ and $v_R \in \mathcal{V}'_R$. Then n_i is not cut in Π', since all pins of n_i reside in $\mathcal{V}'_R$. On the other hand, n'_i is cut in Π', since it connects both $\mathcal{V}'_L$ and $\mathcal{V}'_R$, respectively due to pins $v_L \in \mathcal{V}'_L$ and $v_i \in \mathcal{V}'_R$.

Since there exist two cut nets in Π' for each left-cut net in Π, and one cut net in Π' for each other net in $\mathcal{H}$, the cutsize of Π' is equal to the left-cut-net metric in Π plus the number of nets in $\mathcal{H}$, that is,

$$cutsize(\Pi') = left\text{-}cut\text{-}net(\Pi) + |N|.$$

Since $|\mathcal{N}|$ is fixed, minimizing the cutsize of Π' (1) corresponds to minimizing the left-cut-net metric in Π (5). ◻

Figure 7 displays hypergraph $\mathcal{H}_{5:8}$ given in Figure 4, its extended hypergraph $\mathcal{H}'_{5:8}$, a bipartition Π' of $\mathcal{H}'_{5:8}$, and the bipartition Π of $\mathcal{H}_{5:8}$ induced by Π'. In this figure, the left-cut nets in Π and their corresponding cut nets in Π' are shown in a gray background. Observe that $cutsize(\Pi') = 6$, where $left\text{-}cut\text{-}net(\Pi) = 2$ and $|\mathcal{N}| = 4$, hence, $cutsize(\Pi') = |N| + left\text{-}cut\text{-}net(\Pi)$.

5. Experiments. In this section, we provide the implementation details of the proposed moHP algorithm and the experimental results that compare the performance of the moHP algorithm against those of the state-of-the-art profile reduction algorithms on an extensive dataset. Our experiments are three-fold:

- sensitivity-analysis experiments that compare six different parameter settings for the moHP algorithm in terms of profile and runtime (section 5.3),
- experiments that compare the moHP algorithm against three baseline algorithms in terms of profile and runtime (section 5.4), and
- experiments that compare the moHP algorithm against the best baseline algorithm in terms of the factorization performance in a direct sparse solver (section 5.5).

All experiments are conducted on a Linux workstation equipped with four 18-core CPUs (Intel Xeon Processor E7-8860 v4) and 256 GB of memory.

5.1. Implementation. Recall that in the proposed moHP algorithm, recursion stops when the current hypergraph contains exactly one free vertex. However in our implementation, we allow the flexibility of early stopping when the number of free vertices in the current hypergraph is smaller than or equal to a threshold, which is denoted by t. We refer to this scheme as *early stopping*. Note that early stopping with $t = 1$ is equivalent to the original moHP algorithm. Using $t > 1$ results in an ordered partition with multiple vertices in each part. This partition induces a partial permutation on the rows/columns of the input matrix in such a way that the rows/columns corresponding to the vertices in part $\mathcal{V}_k$ are ordered before those corresponding to the vertices in part $\mathcal{V}_{k+1}$ and after those corresponding to the vertices in part $\mathcal{V}_{k-1}$. In order to determine the internal orderings of the resulting row/column blocks, we adapt and use the weighted greed heuristic proposed for profile reduction in [27]. In the original version of this heuristic, a row/column which maximizes a weight function is selected at each iteration and ordered in the right/bottom of the matrix. In our algorithm, we run this heuristic once for each row/column block so that the selection only considers the rows/columns inside the corresponding block.

The motivation for early stopping is that the quality of the bipartitions obtained by multi-level partitioning tools on very small hypergraphs may not always worth the total runtime of these many bipartitionings on small hypergraphs. Early stopping enables us to exploit the trade-off between the quality and the runtime of the proposed algorithm. Note that the early-stopping scheme with $t = \alpha$ saves at least $\log \alpha$ recursion levels from incurring bipartitioning overhead while losing the merit of performing these unrealized recursion levels. Hence, using a larger threshold results in a faster reordering with a larger profile. The experimental results that compare the performance of the moHP algorithm for varying threshold values are given in section 5.3.

Since the ultimate goal of the proposed model is to obtain an ordering rather than a balanced partitioning, we use a loose balance constraint, i.e., a large ϵ value in (2), in the bipartitionings performed in the proposed algorithm. Using a looser constraint widens the solution space, hence is likely to result in a better quality. The experimental results that compare the performance of the moHP algorithm for varying ϵ values are given in section 5.3.

The proposed algorithm is implemented in C and compiled using gcc version 4.9.2 with optimization level two. All source code is available for download[1]. In each bipartitioning step, PaToH is used as the hypergraph partitioner. In the preliminary experiments, we observed that the performance of the proposed algorithm varies with the parameters of PaToH (see manual [11]) and using Sweep (the vertex visit order), Absorption Matching (the coarsening algorithm) and Kernihgan-Lin (the refinement

[1] https://github.com/seheracer/profilereduction

algorithm) generally gives a better result. Note that the extended hypergraph $\mathcal{H}'$ is obtained from each hypergraph $\mathcal{H}$ to be bipartitioned in line 4 of Algorithm 2. In our efficient implementation, the FORM algorithm obtains the extended hypergraphs $\mathcal{H}'_L$ and $\mathcal{H}'_R$ directly from the extended hypergraph $\mathcal{H}'$, instead of first forming $\mathcal{H}_L$ and $\mathcal{H}_R$ and then obtaining $\mathcal{H}'_L$ and $\mathcal{H}'_R$.

5.2. Dataset. The experiments are conducted on an extensive dataset of symmetric matrices obtained from the SuiteSparse (formerly known as UFL) Sparse Matrix Collection [13]. This dataset is formed by merging the following sets of matrices:

- 131 matrices that are used in the well-known profile reduction works. Since these works were published some time ago, some of these matrices are small in today's standards. These matrices are all symmetric and include:
  - the 18 matrices in Kumfert and Pothen's Collection, which is used in [4, 8, 29, 30, 35, 36],
  - the 8 matrices in NASA Collection, which is used in [4, 27],
  - the 44 AA^T matrices[2] with more than 1000 rows in Netlib Linear Programming Problem Collection, which is used in [27],
  - the 71 matrices with more than 1000 rows in Harwell-Boeing Collection, which is used in [4, 27, 29].
- 176 symmetric matrices in SuiteSparse Collection with the number of nonzeros between 1,000,000 and 100,000,000, excluding the ones whose problem kind is "graph".

Duplicate matrices are excluded from the dataset, that is, only one of the matrices with the same sparsity pattern is kept in the dataset. An error is encountered when HSL code MA67, which is included in the tested baseline algorithms, is run on eight matrices (`boyd1`, `c-73`, `boyd2`, `lp1`, `c-big`, `ins2`, `TSOPF_FS_b39_c30` and `mip1`), hence those eight matrices are excluded from the dataset as well. The resulting dataset[3] consists of 295 matrices.

5.3. Sensitivity analysis. In this section, we analyze the effects of the following parameters (mentioned in section 5.1) on the resulting profile and the runtime of the moHP algorithm:

- t: threshold value for early stopping and
- ϵ: maximum imbalance ratio allowed in each bipartitioning (2). Note that $0 \leq \epsilon \leq 1$ for a bipartition.

We test four different t values (1, 25, 250, and 2500) and two different ϵ values (0.50 and 0.90), hence the number of compared parameter settings is eight. These experiments are conducted on the dataset of 295 matrices described in section 5.2.

Figure 8 displays two performance profile plots [17] which compare the eight different parameter settings for the moHP algorithm. In these plots, label $\epsilon A - tB$ refers to using $\epsilon = A$ and $t = B$. The plot in the left compares these eight settings in terms of profile, whereas the plot in the right compares them in terms of runtime (of the moHP algorithm). Since we apply the weighted greed heuristic [27] for determining the internal ordering of each row/column block for $t > 1$ as mentioned in section 4.5, the runtime values include the runtime of that heuristic as well.

In a performance profile plot [17], the line associated to a method a passing through a point (τ, f) means that in $100f\%$ of the instances, the result obtained by a is at most τ times worse than the best result obtained by the compared methods on

[2] AA^T is performed using MATLAB.

[3] https://github.com/seheracer/profilereduction/blob/master/dataset

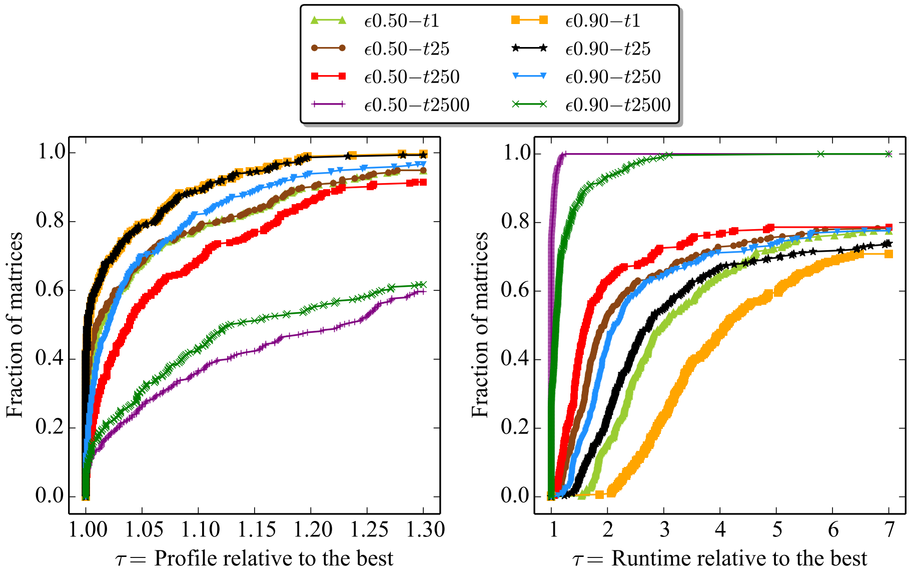

Fig. 8: Performance profile plots comparing the eight versions of the moHP algorithm in terms of profile and runtime.

the corresponding instance. So, the upper a line is, the better the method associated to that line performs.

In Figure 8, the plot in the left shows that $\epsilon 0.90-t25$ and $\epsilon 0.90-t1$ perform the same and outperform the other parameter settings in terms of profile. Observe that for a fixed ϵ value, using a smaller t value improves profile except for going from $t = 25$ to $t = 1$. As the t value decreases, the rate of improvement in profile also decreases and converges to zero for $t = 1$. This finding is in agreement with the motivation of the early stopping scheme described in section 5.1. Also observe that for a fixed t value, $\epsilon = 0.90$ performs better than $\epsilon = 0.50$. This can be attributed to the fact that using $\epsilon = 0.90$ poses a looser constraint compared to using $\epsilon = 0.50$, hence has a larger solution space, as mentioned in section 5.1. Although we only present the results for $\epsilon = 0.50$ and $\epsilon = 0.90$, we also tried using $\epsilon = 0.70$. Expectedly, the performance of $\epsilon = 0.70$ is better than that of $\epsilon = 0.50$ but worse than that of $\epsilon = 0.90$.

In Figure 8, the plot in the right shows that $\epsilon 0.50-t2500$ is the fastest setting, whereas $\epsilon 0.90-t1$ is the slowest one. Observe that using a smaller t value always increases the runtime of the moHP algorithm due to the reasons explained in section 5.1. Using a larger ϵ value also increases the runtime, which can be attributed to the enlargened solution space again.

Considering both of these parameters, one consistent finding is that the runtime of the moHP algorithm increases as the resulting profile decreases. In the experiments given in sections 5.4 and 5.5, we use $\epsilon 0.90$–$t25$ because it is one of the best performers along with $\epsilon 0.90-t1$ in terms of profile and it is considerably faster than $\epsilon 0.90-t1$.

5.4. Comparison against baseline algorithms. In this section, we compare the performance of the moHP algorithm against those of four baseline algorithms, each of which consists of two phases. The heuristics used in these baseline algorithms constitute the state of the art in this field, as also confirmed by [5, 25]. In the first phase of our baseline algorithms, we use one of the following heuristics: *RCM* [21],

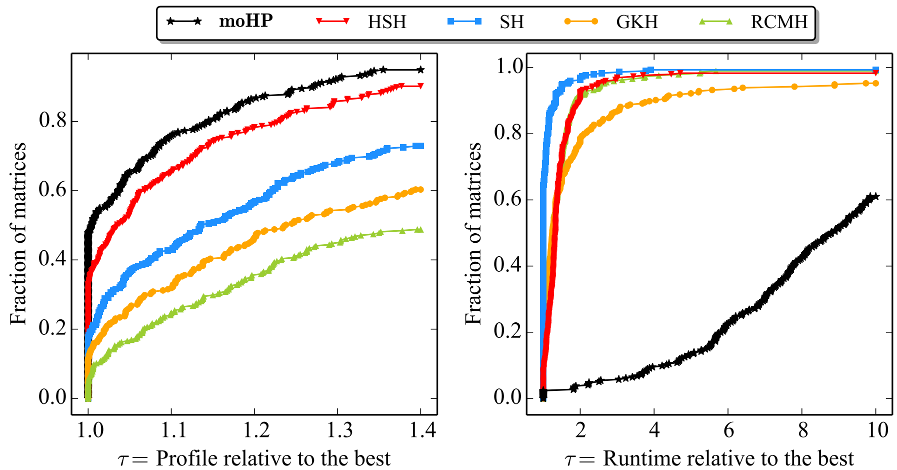

Fig. 9: Performance profile plots comparing the moHP algorithm and the baseline algorithms in terms of profile and runtime.

GibbsKing [22], *Sloan* [40], and *HuScott* [29]. For *RCM*, we use the implementation provided by Reid and Scott [35] in HSL code `MC60` [1]. For *GibbsKing*, we use the efficient implementation provided by Lewis [31] in ACM Algorithm `582`. For *Sloan*, we use the enhanced Sloan algorithm provided by Reid and Scott [35] in HSL code `MC60` [1]. For *HuScott*, we use the multilevel hybrid algorithm provided by Hu and Scott [29] in HSL code `MC73` [1]. In the second phase of each baseline algorithm, we use *Hager*'s exchange algorithm [27] provided by Reid and Scott [36] in HSL code `MC67` [1], because Reid and Scott [36] report that applying Hager's exchange algorithm as a second phase to certain profile reduction algorithms yields better results than using them separately. Then, the baseline algorithms against which we compare the proposed moHP algorithm are summarized as follows:

- *RCMH (RCM+Hager)*: MC60 with `JCNTL(1)=1` followed by MC67.
- *GKH (GibbsKing+Hager)*: The ACM Algorithm 582 followed by MC67.
- *SH (Sloan+Hager)*: MC60 with `JCNTL(1)=0` followed by MC67.
- *HSH (HuScott+Hager)*: MC73 followed by MC67.

Each of these codes is used with default setting and compiled with `gfortran` version `4.9.2` with the `-O2` optimization flag. The double-precision versions are used for the HSL codes.

Figure 9 displays two performance profile plots comparing the proposed moHP algorithm against the baseline algorithms on the dataset of 295 matrices described in section 5.2. Similar to Figure 8, the one in the left compares them in terms of profile, whereas the one in the right compares them in terms of runtime. As seen in the plot in the left, moHP performs significantly better than each baseline algorithm in terms of profile. This can be attributed to the correct formulation of the profile minimization problem as an moHP problem as well as the solution of this problem via recursive bipartitioning utilizing the successful hypergraph partitioning tool PaToH [11]. Among the baseline algorithms, HSH outperforms the rest and is followed by SH and GKH in order. The plot in the right shows that SH is the fastest algorithm, followed by HSH and GKH in order. The moHP algorithm, on the other hand, is the slowest algorithm, which can be explained with the expensive nature of hypergraph partitioning. As will be seen in section 5.5, the quality of the orderings obtained by the moHP algorithm may justify the runtime of the moHP algorithm.

Figure 10 displays eight performance profile plots comparing the proposed moHP algorithm against the baseline algorithms in terms of profile, one for each problem kind having at least ten matrices in our dataset of 295 matrices. The title of each plot displays the respective problem kind and the number of matrices with that kind in parentheses. As seen in the figure, except for kinds `2D/3D` and `Structural`, moHP algorithm performs significantly better than the baseline algorithms. In those problem kinds, moHP is usually followed by HSH, SH, GKH and RCMH in order. For kind `Structural`, moHP and HSH performs comparable, followed by SH, GKH and RCMH in order. For kind `2D/3D`, HSH performs better than all compared algorithms, followed by moHP, SH, GKH and RCMH in order.

5.5. Factorization experiments. In this section, we compare the moHP algorithm only against HSH, which achieves the smallest profile among the baseline algorithms on the average. For the evaluation, in addition to the profile and the ordering runtime, we also consider the factorization performance in a sparse solver, HSL code MA57 [1, 18]. MA57 solves sparse symmetric system(s) of linear equations by using a direct multifrontal method, which is based on a sparse variant of Gaussian elimination. We run MA57 on the matrices reordered by HSH and moHP with default settings and the ordering of each given matrix is kept as is by setting `ICNTL(6)=1`. The reader is referred to the manual[4] for the details of MA57. It is compiled with `gfortran` version 4.9.2 and ATLAS BLAS version 3.11.11.

We consider the following performance metrics obtained during the factorization, i.e., MA57BD:

- *storage:* the number of entries in factors (in millions), i.e., `INFO(15)`/10^6.
- *FLOP count:* the number of floating-point operations for the elimination (in billions), i.e., `RINFO(4)`/10^9.
- *runtime:* the runtime of MA57BD (in seconds).

We perform the MA57 experiments on a dataset containing only large matrices, derived from the dataset given in section 5.2. First, we included all matrices in the initial dataset with number of rows between 100,000 and 500,000. Then, we excluded each matrix whose factorization (MA57BD) takes longer than six hours when the subject matrix is reordered by HSH. The resulting dataset contains 32 matrices whose numbers of nonzeros range between 1,423,116 and 32,886,208. The properties of those matrices and the performance results obtained on them are given in Table 1. In this table, the matrices are sorted in the increasing order of the profile obtained by HSH.

Table 1 displays the properties of the 32 test matrices and the results obtained by HSH and moHP on these matrices. Columns 1, 2 and 3 respectively display the matrix name, the number of rows/columns (m) and the number of nonzeros (nnz). Columns 4-7 display the ordering results, whereas columns 8-13 display the MA57 results. Column pairs 4-5 and 6-7 respectively denote profile and ordering runtime. Column pairs 8-9, 10-11 and 12-13 respectively denote storage, FLOP count and runtime of MA57BD. In each column pair, we compare the performances of HSH and moHP in the respective metric and show the better result in boldface on each matrix. Note that column pair 6-7 displays the runtime of the ordering algorithm, whereas column pair 12-13 displays the runtime of the factorization when the respective matrix is reordered by the corresponding algorithm.

As seen in Table 1, HSH performs better than moHP in terms of profile on matrices

[4] http://www.hsl.rl.ac.uk/specs/ma57.pdf

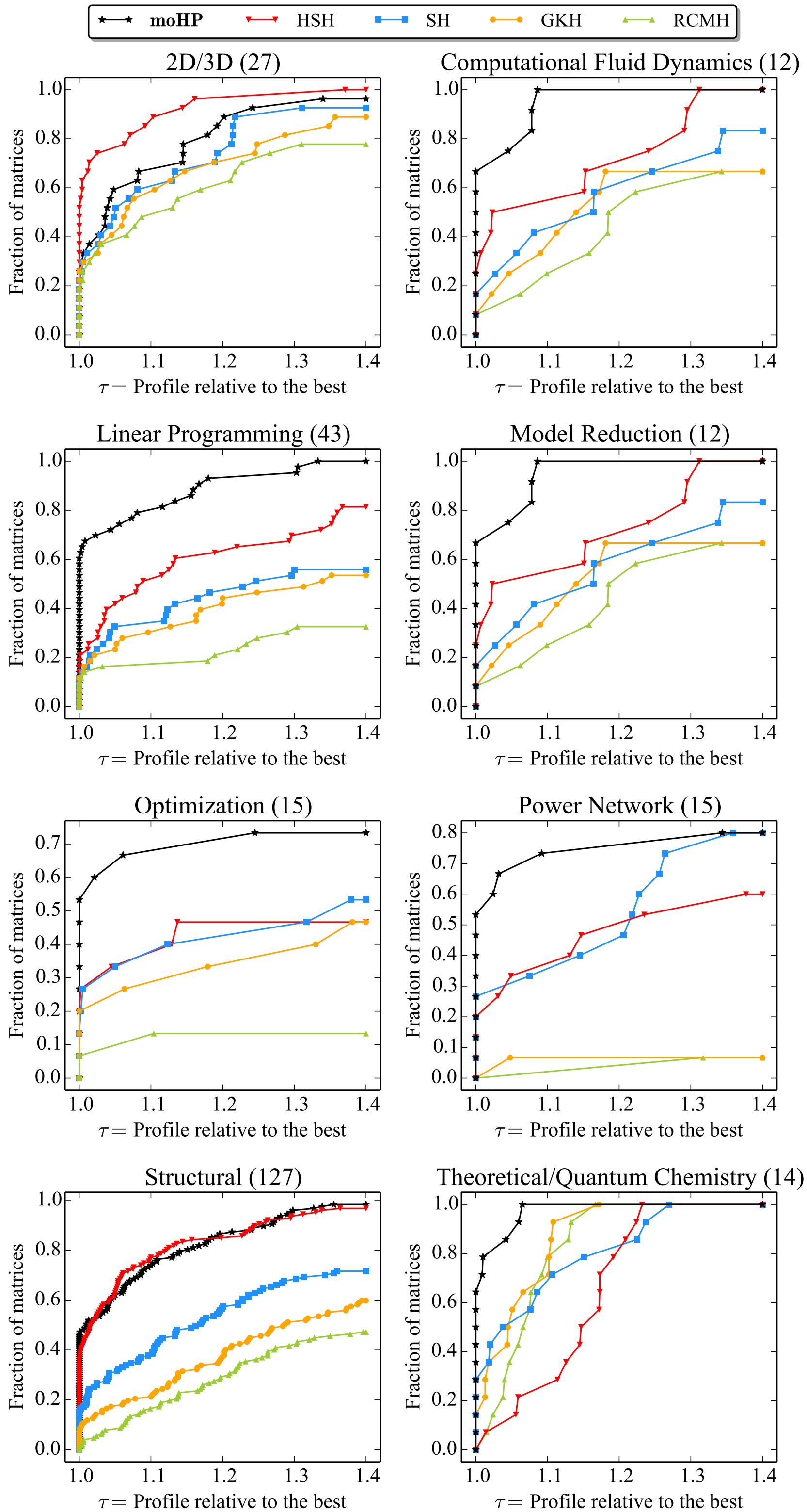

Fig. 10: Performance profile plots comparing the moHP algorithm and the baseline algorithms in terms of profile for different problem kinds. $(\cdot)$ denotes the number of matrices in the respective problem kind.

with small profile, i.e., those on which HSH obtains profile smaller than 230×10^6, except for matrices `filter3D`, `d_pretok` and `2cubes_sphere`. In this set of matrices with small profile, although moHP obtains larger profile than HSH on `bmwcra_1` and comparable profile on `shipsec8` and `shipsec1`, it obtains smaller MA57BD runtime than HSH on these matrices. On all the matrices with large profile, i.e., those on which HSH obtains profile larger than 230×10^6, moHP performs better than HSH except for `Lin`.

As seen in Table 1, HSH runs faster than moHP on all matrices except for `Lin`. However, when we consider the total runtime, which can be expressed as the sum of the ordering and factorization runtimes, moHP performs better than HSH on the matrices with large profile except for `Lin`. For example, consider the largest matrix among those given in Table 1, which is `dielFilterV3clx` with 420,408 rows/columns and 32,886,208 nonzeros. The ordering runtime of moHP on this largest matrix is 102.3 seconds, which is the highest ordering runtime of the moHP algorithm in the given dataset. Even on this matrix, the total runtimes of HSH and moHP respectively are $11.9 + 953.3 = 965.2$ seconds and $102.3 + 743.3 = 845.6$ seconds, so, moHP performs $965.2/845.6 = 1.14$x better than HSH in terms of the total runtime. Similarly, consider the matrix with the largest profile, which is `SiO2` with 155,331 rows/columns and 11,283,503 nonzeros. On this matrix, the total runtimes of HSH and moHP respectively are $11.0 + 20{,}719.7 = 20{,}730.7$ seconds and $62.7 + 15{,}324.9 = 15{,}387.6$ seconds, so, moHP performs $20{,}730.7/15{,}387.6 = 1.35$x better than HSH in terms of the total runtime. Hence, for the matrices with large profile, the better but slower orderings obtained by the moHP algorithm generally pay off very well since they significantly reduce the factorizaton runtimes.

Table 1: Performance comparison of HSH and moHP in terms of profile, ordering runtime and MA57BD's storage, FLOP count and runtime.

matrix properties			ordering				MA57BD					
			profile (10^6)		runtime (s)		storage (10^6)		FLOP count (10^9)		runtime (s)	
name	m	nnz	HSH	moHP	HSH	moHP	HSH	moHP	HSH	moHP	HSH	moHP
`thermomech_dM`	204,316	1,423,116	**28.7**	32.3	**1.5**	18.6	**30.4**	34.0	**4.8**	5.9	**4.1**	5.7
`Dubcova3`	146,689	3,636,649	**60.6**	69.4	**1.4**	13.6	**61.4**	68.6	**29.1**	36.4	**18.7**	24.3
`filter3D`	106,437	2,707,179	65.6	**52.0**	**1.5**	14.7	66.5	**51.6**	46.9	**27.7**	31.6	**19.3**
`darcy003`	389,874	2,101,242	**94.8**	108.5	**2.5**	31.2	**98.1**	111.5	**28.2**	36.7	**20.6**	26.2
`d_pretok`	182,730	1,641,672	94.9	**94.5**	**1.3**	17.5	96.6	**96.1**	58.4	**56.8**	40.3	**39.1**
`bmw7st_1`	141,347	7,339,667	**106.1**	133.3	**2.5**	19.9	**101.9**	116.9	**84.3**	116.6	**55.6**	75.4
`turon_m`	189,924	1,690,876	**113.1**	113.7	**1.8**	18.4	**114.8**	115.1	**72.9**	76.0	**50.3**	51.9
`cfd2`	123,440	3,087,898	**131.0**	136.9	**1.5**	17.5	**132.1**	136.9	**149.3**	173.2	**98.9**	113.7
`hood`	220,542	10,768,436	**139.2**	164.9	**3.4**	30.3	**140.9**	147.5	**100.6**	106.2	**61.6**	64.2
`BenElechi1`	245,874	13,150,496	**152.7**	181.4	**3.7**	31.7	**154.4**	171.1	**102.9**	135.3	**70.3**	90.4
`2cubes_sphere`	101,492	1,647,264	154.9	**143.8**	**1.2**	14.7	155.7	**144.0**	264.4	**229.0**	177.7	**151.6**
`pwtk`	217,918	11,634,424	**159.3**	176.5	**4.2**	26.2	**159.7**	167.6	**119.3**	135.4	**80.3**	89.5
`bmwcra_1`	148,770	10,644,002	**159.9**	182.3	**3.4**	32.8	161.1	**140.7**	198.2	**149.8**	129.1	**97.3**
`ship_003`	121,728	8,086,034	**164.6**	193.8	**2.3**	19.7	**152.5**	167.4	**217.4**	298.0	**136.9**	192.1
`shipsec8`	114,919	6,653,399	**180.5**	181.9	**2.0**	16.8	174.7	**169.2**	299.7	**292.3**	192.1	**184.7**
`helm2d03`	392,257	2,741,935	**194.7**	201.8	**8.9**	33.9	**198.1**	205.1	**114.1**	122.1	**78.2**	83.7
`shipsec1`	140,874	7,813,404	**203.1**	209.7	**2.4**	20.2	198.7	**189.0**	315.9	**302.9**	203.0	**193.0**
`shipsec5`	179,860	10,113,096	**229.6**	304.8	**3.3**	25.0	**228.8**	253.6	**304.3**	463.2	**193.2**	301.0
`boneS01`	127,224	6,715,152	245.5	**226.5**	**2.2**	20.5	245.5	**219.5**	549.1	**444.3**	366.4	**289.1**
`bmw3_2`	227,362	11,288,630	285.9	**272.5**	**4.7**	32.0	278.6	**253.9**	400.3	**349.0**	260.5	**222.5**
`wave`	156,317	2,118,662	293.0	**265.7**	**2.2**	21.3	294.3	**266.4**	640.8	**519.0**	477.5	**371.9**
`CurlCurl_1`	226,451	2,472,071	414.4	**380.6**	**1.4**	27.7	416.2	**361.6**	957.5	**708.5**	718.5	**493.3**
`msdoor`	415,863	20,240,935	416.5	**393.9**	**6.9**	59.7	419.7	**367.4**	461.3	**349.5**	280.3	**212.2**
`offshore`	259,789	4,242,673	516.9	**388.8**	**3.2**	40.2	518.9	**377.4**	1,171.9	**656.3**	862.5	**446.4**
`Lin`	256,000	1,766,400	**544.6**	585.6	106.5	**29.1**	**546.8**	587.8	**1,317.8**	1,540.6	**947.6**	1,129.9
`F1`	343,791	26,837,113	**592.9**	652.2	**12.9**	90.9	594.6	**551.4**	1,209.6	**1,066.8**	793.2	**689.1**
`dielFilterV3clx`	420,408	32,886,208	731.0	**698.7**	**11.9**	102.3	730.6	**675.1**	1,460.9	**1,194.2**	953.3	**743.3**
`Ge99H100`	112,985	8,451,395	1,144.9	**960.6**	**8.6**	48.4	1,145.7	**961.5**	13,242.9	**9,004.0**	10,152.1	**6,759.9**
`Ga10As10H30`	113,081	6,115,633	1,157.2	**1,018.0**	**5.6**	45.0	1,158.0	**1,018.9**	13,819.8	**10,280.9**	10,527.5	**7,762.**1
`Ge87H76`	112,985	7,892,195	1,169.8	**955.7**	**7.7**	46.7	1,170.6	**956.5**	13,981.3	**8,907.3**	10,785.8	**6,713.3**
`Ga19As19H42`	133,123	8,884,839	1,523.7	**1,311.5**	**10.2**	59.6	1,524.7	**1,312.6**	20,166.4	**14,362.3**	15,681.2	**11,124.0**
`SiO2`	155,331	11,283,503	1,910.2	**1,695.9**	**11.0**	62.7	1,911.5	**1,684.3**	26,578.2	**20,036.9**	20,719.7	**15,324.9**

6. Conclusion. We formulated the profile minimization problem as a constrained version of the hypergraph partitioning (HP) problem, which we refer to as the m-way ordered hypergraph partitioning (moHP) problem. For solving the moHP problem, we proposed the moHP algorithm, which utilizes the recursive bipartitioning approach. The moHP algorithm addresses the minimization objective of the moHP problem by utilizing fixed vertices and two novel cut-net manipulation techniques. We theoretically showed the correctness of the proposed moHP algorithm and described how the existing partitioning tools can be utilized in the moHP algorithm. We tested the performance of the moHP algorithm against the state-of-the-art profile reduction algorithms on a large dataset of 295 matrices and the experimental results showed the validity of the proposed approach.

Appendix A. Proofs of lemmas. We prove each of Lemmas 3, 4, 5, and 6 by induction on the depth of the RB tree. We assume that the depth of $\mathcal{T}^{\mathcal{H}}$ is $k > 0$. One important observation is that the depths of subtrees $\mathcal{T}^{\mathcal{H}_L}$ and $\mathcal{T}^{\mathcal{H}_R}$ are both less than k. The base case for the induction in each proof corresponds to the depth of $\mathcal{T}^{\mathcal{H}}$ being equal to zero, which implies that $\mathcal{H}$ is represented by a leaf node. In this case, no further moHP invocations are carried on, hence, $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = 0$.

In the following proofs, we use $\hat{\mathcal{V}}$, $\hat{\mathcal{V}}_L$, and $\hat{\mathcal{V}}_R$ to denote the number of free vertices in $\mathcal{H}$, $\mathcal{H}_L$, and $\mathcal{H}_R$, respectively.

A.1. Lemma 3.

Proof. In the base case, $\hat{\mathcal{V}} - 1 = 0$ since there is exactly one free vertex in $\mathcal{H}$, hence, $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\hat{\mathcal{V}}| - 1$ holds.

We assume $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\hat{\mathcal{V}}| - 1$ holds when the tree depth is less than k. Since n_i is left-right-anchored, it is left-cut in the bipartition of $\mathcal{H}$. Thus, n_i is copied to both $\mathcal{H}_L$ and $\mathcal{H}_R$ by net duplication technique and it is left-right-anchored in both of them. By the inductive hypothesis, $\mu(n_i, \mathcal{T}^{\mathcal{H}_L}) = |\hat{\mathcal{V}}_L| - 1$ and $\mu(n_i, \mathcal{T}^{\mathcal{H}_R}) = |\hat{\mathcal{V}}_R| - 1$. Notice that $\hat{\mathcal{V}}_L$ and $\hat{\mathcal{V}}_R$ are disjoint and $\hat{\mathcal{V}} = \hat{\mathcal{V}}_L \cup \hat{\mathcal{V}}_R$. Since n_i is left-cut in the bipartition of $\mathcal{H}$, we have

$$\begin{aligned}\mu(n_i, \mathcal{T}^{\mathcal{H}}) &= \mu(n_i, \mathcal{T}^{\mathcal{H}_L}) + \mu(n_i, \mathcal{T}^{\mathcal{H}_R}) + 1 = (|\hat{\mathcal{V}}_L| - 1) + (|\hat{\mathcal{V}}_R| - 1) + 1 \\ &= |\hat{\mathcal{V}}_L| + |\hat{\mathcal{V}}_R| - 1 = |\hat{\mathcal{V}}| - 1.\end{aligned}$$

∎

A.2. Lemma 4.

Proof. Recall that n_i connects f_i in $\mathcal{H}$ since it is right-anchored. In the base case, $\hat{\mathcal{V}} = \{f_i\}$ since there is exactly one free vertex in $\mathcal{H}$. Then, $|\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}| = 0$, hence, $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}|$ holds.

We assume $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}|$ holds when the tree depth is less than k. We investigate the cases of n_i being cut or not in the bipartition of $\mathcal{H}$ as follows.

1. *n_i is cut:* Since n_i is right-anchored, it is left-cut. Then, n_i is copied to both $\mathcal{H}_L$ and $\mathcal{H}_R$ by net duplication technique and it is right-anchored and left-right-anchored in $\mathcal{H}_L$ and $\mathcal{H}_R$, respectively. By the inductive hypothesis, $\mu(n_i, \mathcal{T}^{\mathcal{H}_L}) = |\{v \in \hat{\mathcal{V}}_L : \phi(v) > \phi(f_i)\}|$. Moreover, by Lemma 3, $\mu(n_i, \mathcal{T}^{\mathcal{H}_R}) = |\hat{\mathcal{V}}_R| - 1$. Notice that each vertex in $\hat{\mathcal{V}}_R$ is numbered after f_i

since $f_i \in \hat{\mathcal{V}}_L$. Since n_i is left-cut in the bipartition of $\mathcal{H}$, we have

$$\begin{aligned}
\mu(n_i, \mathcal{T}^{\mathcal{H}}) &= \mu(n_i, \mathcal{T}^{\mathcal{H}_L}) + \mu(n_i, \mathcal{T}^{\mathcal{H}_R}) + 1 \\
&= |\{v \in \hat{\mathcal{V}}_L : \phi(v) > \phi(f_i)\}| + (|\hat{\mathcal{V}}_R| - 1) + 1 \\
&= |\{v \in \hat{\mathcal{V}}_L : \phi(v) > \phi(f_i)\}| + |\hat{\mathcal{V}}_R| = |\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}|.
\end{aligned}$$

2. *n_i is not cut:* Since n_i is right-anchored, it is internal to the right part, which implies that it appears only in $\mathcal{H}_R$. Then, n_i is right-anchored in $\mathcal{H}_R$. Then, by inductive hypothesis,

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = \mu(n_i, \mathcal{T}^{\mathcal{H}_R}) = |\{v \in \hat{\mathcal{V}}_R : \phi(v) > \phi(f_i)\}| = |\{v \in \hat{\mathcal{V}} : \phi(v) > \phi(f_i)\}|.$$

□

A.3. Lemma 5.

Proof. Recall that n_i connects f_i in $\mathcal{H}$ since it is left-anchored. In the base case, $\hat{\mathcal{V}} = \{v_i\}$ since there is exactly one free vertex. Then, $|\{v \in \hat{\mathcal{V}} : \phi(v) < \phi(v_i)\}| = 0$, hence, $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) < \phi(v_i)\}|$ holds.

We assume $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(v) < \phi(v_i)\}|$ holds when the depth is less than k. We investigate the cases of n_i being left-cut or not in the bipartition of $\mathcal{H}$ as follows.

1. *n_i is left-cut.* n_i is copied to both $\mathcal{H}_L$ and $\mathcal{H}_R$ by net duplication and it is left-right-anchored and left-anchored at $\mathcal{H}_L$ and $\mathcal{H}_R$, respectively. By the inductive hypothesis, $\mu(n_i, \mathcal{T}_R^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}|$. Moreover, by Lemma 3, $\mu(n_i, \mathcal{T}_L^{\mathcal{H}}) = |\hat{\mathcal{V}}| - 1$. Notice that each vertex in $\hat{\mathcal{V}}_R$ is numbered before v_i since $v_i \in \hat{\mathcal{V}}_R$. Since n_i is left-cut in the bipartition of $\mathcal{H}$, we have

$$\begin{aligned}
\mu(n_i, \mathcal{T}^{\mathcal{H}}) &= \mu(n_i, \mathcal{T}^{\mathcal{H}_L}) + \mu(n_i, \mathcal{T}^{\mathcal{H}^R}) + 1 \\
&= (|\hat{\mathcal{V}}_L| - 1) + |\{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}| + 1 \\
&= |\hat{\mathcal{V}}_L| + \{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}| = |\{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}|.
\end{aligned}$$

2. *n_i is not left-cut:* Since n_i is left-anchored, n_i appears only in $\mathcal{H}_L$. Then, n_i is left-anchored in $\mathcal{H}_L$ and a vertex $v \in \hat{\mathcal{V}}$ is in $\hat{\mathcal{V}}_L$ whenever $\phi(v) < \phi(v_i)$. Then, by the inductive hypothesis,

$$\mu(n_i, \mathcal{T}^{\mathcal{H}}) = \mu(n_i, \mathcal{T}_L^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}}_L : \phi(v) < \phi(v_i)\}| = |\{v \in \hat{\mathcal{V}} : \phi(v) < \phi(v_i)\}|.$$

□

A.4. Lemma 6.

Proof. Recall that both f_i and v_i are free vertices and connected by n_i in $\mathcal{H}$ since n_i is free. In the base case, $\hat{\mathcal{V}} = \{f_i = v_i\}$ since there is exactly one free vertex. Then, $|\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1 = 0$, hence, $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}|$.

We assume $\mu(n_i, \mathcal{T}^{\mathcal{H}}) = |\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}|$ holds when the depth is less than k. We investigate the cases of n_i being left-cut or not in the bipartition of $\mathcal{H}$ as follows.

1. *n_i is left-cut:* n_i is copied to both $\mathcal{H}_L$ and $\mathcal{H}_R$ by net duplication and it is right-anchored and left-anchored in $\mathcal{H}_L$ and $\mathcal{H}_R$, respectively. By Lemma 4, $\mu(n_i, \mathcal{T}^{\mathcal{H}_L}) = |\{v \in \hat{\mathcal{V}}_L : \phi(v) > \phi(f_i)\}|$. By Lemma 5, $\mu(n_i, \mathcal{T}^{\mathcal{H}_R}) = |\{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}|$. Notice that each vertex in $\hat{\mathcal{V}}_L$ is numbered before v_i

since $v_i \in \hat{\mathcal{V}}_R$. Also notice that each vertex in $\hat{\mathcal{V}}_R$ is numbered after f_i since $f_i \in \hat{\mathcal{V}}_L$. Since n_i is left-cut in the bipartition of $\mathcal{H}$, we have

$$\begin{aligned}
\mu(n_i, \mathcal{T}^{\mathcal{H}}) &= \mu(n_i, \mathcal{T}_L^{\mathcal{H}}) + \mu(n_i, \mathcal{T}_R^{\mathcal{H}}) + 1 \\
&= |\{v \in \hat{\mathcal{V}}_L : \phi(v) > \phi(f_i)\}| + |\{v \in \hat{\mathcal{V}}_R : \phi(v) < \phi(v_i)\}| + 1 \\
&= (|\{v \in \hat{\mathcal{V}}_L : \phi(v) \geq \phi(f_i)\}| - 1) + (|\{v \in \hat{\mathcal{V}}_R : \phi(v) \leq \phi(v_i)\}| - 1) + 1 \\
&= |\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1.
\end{aligned}$$

2. n_i *is not left-cut:* Since n_i is free in $\mathcal{H}$, n_i appears in only one of $\mathcal{H}_L$ and $\mathcal{H}_R$ wherein n_i remains to be free. Consider a vertex $v \in \hat{\mathcal{V}}$ satisfying $\phi(f_i) \leq \phi(v) \leq \phi(v_i)$. If $n_i \in \mathcal{H}_L$, $v \in \hat{\mathcal{V}}_L$, otherwise, $v \in \hat{\mathcal{V}}_R$. Without loss of generality, assume that n_i appears in only $\mathcal{H}_L$. Then, by the inductive hypothesis,

$$\begin{aligned}
\mu(n_i, \mathcal{T}^{\mathcal{H}}) = \mu(n_i, \mathcal{T}_L^{\mathcal{H}}) &= |\{v \in \hat{\mathcal{V}}_L : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1 \\
&= |\{v \in \hat{\mathcal{V}} : \phi(f_i) \leq \phi(v) \leq \phi(v_i)\}| - 1.
\end{aligned}$$

□

Acknowledgment. We thank the anonymous referees for their valuable comments, which helped us substantially improve the presentation of this paper.

REFERENCES

[1] *HSL. A Collection of Fortran Codes for Large Scale Scientific Computation*, http://www.hsl.rl.ac.uk/.

[2] S. Acer, E. Kayaaslan, and C. Aykanat, *A recursive bipartitioning algorithm for permuting sparse square matrices into block diagonal form with overlap*, SIAM Journal on Scientific Computing, 35 (2013), pp. C99–C121.

[3] S. Acer, O. Selvitopi, and C. Aykanat, *Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems*, Parallel Computing, 59 (2016), pp. 71 – 96, https://doi.org/https://doi.org/10.1016/j.parco.2016.10.001, http://www.sciencedirect.com/science/article/pii/S0167819116301041. Theory and Practice of Irregular Applications.

[4] S. T. Barnard, A. Pothen, and H. D. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, Numerical Linear Algebra with Applications, 2 (1995), pp. 317–334, https://doi.org/10.1002/nla.1680020402, http://dx.doi.org/10.1002/nla.1680020402.

[5] J. A. B. Bernardes and S. L. G. de Oliveira, *A systematic review of heuristics for profile reduction of symmetric matrices*, Procedia Computer Science, 51 (2015), pp. 221 – 230, https://doi.org/https://doi.org/10.1016/j.procs.2015.05.231, http://www.sciencedirect.com/science/article/pii/S187705091501039X. International Conference On Computational Science, ICCS 2015.

[6] M. W. Berry, B. Hendrickson, and P. Raghavan, *Sparse matrix reordering schemes for browsing hypertext*, Lectures in Applied Mathematics-American Mathematical Society, 32 (1996), pp. 99–124.

[7] M. E. Bolanos, S. Aviyente, and H. Radha, *Graph entropy rate minimization and the compressibility of undirected binary graphs*, in 2012 IEEE Statistical Signal Processing Workshop (SSP), Aug 2012, pp. 109–112, https://doi.org/10.1109/SSP.2012.6319634.

[8] E. G. Boman and B. Hendrickson, *A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices*, Tech. Report SCCM-96-14, Stanford University, Stanford, CA, 1996.

[9] D. Burgess and M. Giles, *Renumbering unstructured grids to improve the performance of codes on hierarchical memory machines*, Advances in Engineering Software, 28 (1997), pp. 189 – 201, https://doi.org/https://doi.org/10.1016/S0965-9978(96)00039-7, http://www.sciencedirect.com/science/article/pii/S0965997896000397.

[10] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, Parallel and Distributed Systems, IEEE Transactions on, 10 (1999), pp. 673–693, https://doi.org/10.1109/71.780863.

[11] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*, Dept. of Computer Engineering, Bilkent University, Ankara, 06533 Turkey, 1999. PaToH is available at http://bmi.osu.edu/~umit/software.htm.

[12] S. S. Clift and W.-P. Tang, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, BIT Numerical Mathematics, 35 (1995), pp. 30–47, https://doi.org/10.1007/BF01732977, https://doi.org/10.1007/BF01732977.

[13] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25, http://www.cise.ufl.edu/research/sparse/matrices.

[14] T. A. Davis, S. Rajamanickam, and W. M. Sid-Lakhdar, *A survey of direct methods for sparse linear systems*, Acta Numerica, 25 (2016), p. 383566, https://doi.org/10.1017/S0962492916000076.

[15] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 944–961, https://doi.org/10.1137/0613057, https://doi.org/10.1137/0613057, https://arxiv.org/abs/https://doi.org/10.1137/0613057.

[16] J. Díaz, J. Petit, and M. Serna, *A survey of graph layout problems*, ACM Comput. Surv., 34 (2002), pp. 313–356, https://doi.org/10.1145/568522.568523, http://doi.acm.org/10.1145/568522.568523.

[17] E. D. Dolan and J. J. Morè, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213, https://doi.org/10.1007/s101070100263, http://dx.doi.org/10.1007/s101070100263.

[18] I. S. Duff, *Ma57—a code for the solution of sparse symmetric definite and indefinite systems*, ACM Trans. Math. Softw., 30 (2004), pp. 118–144, https://doi.org/10.1145/992200.992202, http://doi.acm.org/10.1145/992200.992202.

[19] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT Numerical Mathematics, 29 (1989), pp. 635–657, https://doi.org/10.1007/BF01932738, https://doi.org/10.1007/BF01932738.

[20] C. A. Felippa, *Introduction to finite element methods*, Department of Aerospace Engineering Sciences and Center for Aerospace Structures, University of Colorado Boulder, (2001).

[21] A. George, *Computer implementation of the finite element method*, PhD thesis, Stanford University, Stanford, CA, 1971.

[22] N. E. Gibbs, *Algorithm 509: A hybrid profile reduction algorithm [F1]*, ACM Trans. Math. Softw., 2 (1976), pp. 378–387, https://doi.org/10.1145/355705.355713, http://doi.acm.org/10.1145/355705.355713.

[23] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer, *An algorithm for reducing the bandwidth and profile of a sparse matrix*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 236–250, https://doi.org/10.1137/0713023, http://epubs.siam.org/doi/abs/10.1137/0713023, https://arxiv.org/abs/http://epubs.siam.org/doi/pdf/10.1137/0713023.

[24] S. L. Gonzaga de Oliveira, J. A. B. Bernardes, and G. O. Chagas, *An evaluation of reordering algorithms to reduce the computational cost of the incomplete Cholesky-conjugate gradient method*, Computational and Applied Mathematics, (2017), https://doi.org/10.1007/s40314-017-0490-5, https://doi.org/10.1007/s40314-017-0490-5.

[25] S. L. Gonzaga de Oliveira, J. A. B. Bernardes, and G. O. Chagas, *An evaluation of low-cost heuristics for matrix bandwidth and profile reductions*, Computational and Applied Mathematics, 37 (2018), pp. 1412–1471, https://doi.org/10.1007/s40314-016-0394-9, https://doi.org/10.1007/s40314-016-0394-9.

[26] P. Grindrod, *Range-dependent random graphs and their application to modeling large small-world proteome datasets*, Phys. Rev. E, 66 (2002), p. 066702, https://doi.org/10.1103/PhysRevE.66.066702, https://link.aps.org/doi/10.1103/PhysRevE.66.066702.

[27] W. W. Hager, *Minimizing the profile of a symmetric matrix*, SIAM Journal on Scientific Computing, 23 (2002), pp. 1799–1816.

[28] D. J. Higham, *Unravelling small world networks*, Journal of Computational and Applied Mathematics, 158 (2003), pp. 61 – 74, https://doi.org/https://doi.org/10.1016/S0377-0427(03)00471-0, http://www.sciencedirect.com/science/article/pii/S0377042703004710. Selection of papers from the Conference on Computational and Mathematical Methods for Science and Engineering, Alicante University, Spain, 20-25 September 2002.

[29] Y. Hu and J. Scott, *A multilevel algorithm for wavefront reduction*, SIAM Journal on Scientific Computing, 23 (2001), pp. 1352–1375, https://doi.org/10.1137/S1064827500377733, http://epubs.siam.org/doi/abs/10.1137/S1064827500377733, https://arxiv.org/abs/http://epubs.siam.org/doi/pdf/10.1137/S1064827500377733.

[30] G. Kumfert and A. Pothen, *Two improved algorithms for envelope and wavefront reduction*,

BIT Numerical Mathematics, 37 (1997), pp. 1–32, https://doi.org/10.1007/BF02510240, http://dx.doi.org/10.1007/BF02510240.

[31] J. G. Lewis, *Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms*, ACM Trans. Math. Softw., 8 (1982), pp. 180–189, https://doi.org/10.1145/355993.355998, http://doi.acm.org/10.1145/355993.355998.

[32] Y. Lin and J. Yuan, *Profile minimization problem for matrices and graphs*, Acta Mathematicae Applicatae Sinica, 10 (1994), pp. 107–112, https://doi.org/10.1007/BF02006264, http://dx.doi.org/10.1007/BF02006264.

[33] J. Meijer and J. van de Pol, *Bandwidth and Wavefront Reduction for Static Variable Ordering in Symbolic Reachability Analysis*, Springer International Publishing, Cham, 2016, pp. 255–271, https://doi.org/10.1007/978-3-319-40648-0_20, https://doi.org/10.1007/978-3-319-40648-0_20.

[34] C. Mueller, B. Martin, and A. Lumsdaine, *A comparison of vertex ordering algorithms for large graph visualization*, in 2007 6th International Asia-Pacific Symposium on Visualization, Feb 2007, pp. 141–148, https://doi.org/10.1109/APVIS.2007.329289.

[35] J. K. Reid and J. A. Scott, *Ordering symmetric sparse matrices for small profile and wavefront*, International Journal for Numerical Methods in Engineering, 45 (1999), pp. 1737–1755.

[36] J. K. Reid and J. A. Scott, *Implementing Hager's exchange methods for matrix profile reduction*, ACM Trans. Math. Softw., 28 (2002), pp. 377–391, https://doi.org/10.1145/592843.592844, http://doi.acm.org/10.1145/592843.592844.

[37] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.

[38] O. Selvitopi, S. Acer, and C. Aykanat, *A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously*, IEEE Transactions on Parallel and Distributed Systems, 28 (2017), pp. 345–358, https://doi.org/10.1109/TPDS.2016.2577024.

[39] D. Silva, M. Velazco, and A. Oliveira, *Influence of matrix reordering on the performance of iterative methods for solving linear systems arising from interior point methods for linear programming*, Mathematical Methods of Operations Research, 85 (2017), pp. 97–112, https://doi.org/10.1007/s00186-017-0571-7, https://doi.org/10.1007/s00186-017-0571-7.

[40] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, International Journal for Numerical Methods in Engineering, 23 (1986), pp. 239–251, https://doi.org/10.1002/nme.1620230208, http://dx.doi.org/10.1002/nme.1620230208.

[41] S. Xu, W. Xue, and H. X. Lin, *Performance modeling and optimization of sparse matrix-vector multiplication on nvidia cuda platform*, The Journal of Supercomputing, 63 (2013), pp. 710–721, https://doi.org/10.1007/s11227-011-0626-0, https://doi.org/10.1007/s11227-011-0626-0.