# Reordering Matrices for Optimal Sparse Matrix Bipartitioning

Aras Mumcuyan
Computer Science and Engineering,
Sabancı University, Istanbul, Turkey
arasmumcuyan@sabanciuniv.edu

Kamer Kaya
Computer Science and Engineering,
Sabancı University, Istanbul, Turkey
kaya@sabanciuniv.edu

Hüsnü Yenigün
Computer Science and Engineering,
Sabancı University, Istanbul, Turkey
yenigun@sabanciuniv.edu

***Abstract*—Sparse-matrix vector multiplication (SpMV) is one of the widely used and extensively studied kernels in today's scientific computing and high-performance computing domains. The efficiency and scalability of this kernel is extensively investigated on single-core, multi-core, many-core processors and accelerators, and on distributed memory. In general, a good mapping of an application's tasks to the processing units in a distributed environment is important since communication among these tasks is the main bottleneck on scalability. A fundamental approach to solve this problem is modeling the application via a graph/hypergraph and partitioning it. For SpMV, several graph/hypergraph models have been proposed. These approaches consider the problem as a balanced partitioning problem where the vertices (tasks) are partitioned (assigned) to the parts (processors) in a way that the total vertex weight (processor load) is balanced and the total communication incurred among the processors is minimized.**

**The partitioning problem is NP-Hard and all the existing studies and tools use heuristics to solve the problem. For graphs, the literature on optimal partitioning contains a number of notable studies; however for hypergraphs, very little work has been done. Unfortunately, it has been shown that unlike graphs, hypergraphs can exactly model the total communication for SpMV.**

**Recently, Pelt and Bisseling proposed a novel, purely combinatorial branch-and-bound-based approach for the sparse-matrix bipartitioning problem which can tackle relatively larger hypergraphs that were impossible to optimally partition into two by using previous methods. This work can be considered as an extension to their approach with two ideas. We propose to use; 1) matrix ordering techniques to use more information in the earlier branches of the tree, and 2) a machine learning approach to choose the best ordering based on matrix features. As our experiments on various matrices will show, these enhancements make the optimal bipartitioning process much faster.**

***Index Terms*—Sparse-matrix bipartitioning, hypergraphs, branch-and-bound, matrix ordering, machine learning.**

## I. Introduction

The sparse-matrix vector multiplication (SpMV) is the most time consuming kernel for many applications in scientific computing. SpMV on a distributed setting is a well studied problem, and as for many applications, processors with unequal loads and the communication among the distributed tasks is the main obstacle for scalability. The problem is usually modeled as a graph/hypergraph partitioning problem; the objective is minimizing the number (or total weight) of the cut edges where a small imbalance on the part weights is allowed [1], [2], [3], [4], [5].

For SpMV, unlike graphs, hypergraphs can exactly model the communication [6]. There exist tools in the literature to partition the hypergraphs, e.g., PaToH [7] and Mondriaan [8]; we will also employ these tools. They have been widely used in practice and for research purposes on various problems such as data-intensive computing [9], recommendation systems [10] and iterative sparse-matrix vector multiplication [1], [2], [11]. Although we assume that, and feel by experience, the tools provide high-quality solutions, one cannot evaluate their real performance with respect to the best solution without the optimal results at hand. Some studies and surveys compare the tools; we know their relative performance but we also know that this differs with respect to the application, matrix class, graph type etc. If we had the optimal results, we could identify the weaknesses of these tools, if they exist, in a clear fashion and use them to make the tools better. Unfortunately, the problem is NP-Hard [12]. For graph partitioning, there are numerous studies on optimal (bi)partitioning, bur for hypergraphs, only a few studies exist which are not capable to tackle optimal sparse-matrix bipartitioning problem for matrices with reasonable sizes. A short review of these optimal graph- and hypergraph-partitioning literature can be found in [13].

Recently, based on their medium-grain method [14], Pelt and Bisseling proposed a purely combinatorial, branch-and-bound-based approach that computationally does not explore the nonzeros of the sparse matrix but consider the rows and columns [13]. Since the number of nonzeros is much more than the number of rows and columns this approach reduces the complexity in practice. For this task, they implemented `MondriaanOpt`[1] which is publicly available. Our work is built on their software and uses two ideas: 1) ordering the matrices to use more information in the earlier branches/levels of the tree, and 2) designing a machine learning approach to choose the best ordering based on the matrix features. As our experiments on various matrices will show, these enhancements make the optimal sparse-matrix bipartitioning process much faster.

The rest of the paper is organized is as follows: In Section II, we introduce the sparse-matrix bipartitioning problem. Section III describes the ordering algorithms we used and Section IV introduces the machine learning method we employed. The experimental results are given in Section V. Section VI concludes the paper and discusses possible future work.

[1]http://www.staff.science.uu.nl/ bisse101/Mondriaan/Opt/

## II. Notation and Background

*Sparse-matrix bipartitioning:* Given a sparse matrix $\mathbf{A}$ and a load imbalance parameter $\epsilon$, we want to divide the matrix into two matrices $\mathbf{A} = \mathbf{A}_1 + \mathbf{A}_2$ such that

$$\max(nnz(\mathbf{A}_1), nnz(\mathbf{A}_2)) \leq (1+\epsilon) \left\lceil \frac{N}{2} \right\rceil$$

where $nnz(\mathbf{A})$ denote the number of nonzeros in $\mathbf{A}$. For simplicity and to avoid the possibility of sharing a nonzero in both of the matrices, we ignore the numerical values and only consider the sparsity pattern of $\mathbf{A}$; that is $\mathbf{A}$, $\mathbf{A}_1$ and $\mathbf{A}_2$ can be considered as 0-1 matrices. While doing that, we want to minimize the communication incurred by the partitioning during a parallel iterative SpMV with two processors; one working on $\mathbf{A}_1$ and the other working on $\mathbf{A}_2$. Let us first define the analogy between a matrix and a fine-grain hypergraph for 2D-decomposition [15], [14]; Figure 1 shows an example; the rows and columns of the matrix correspond to the hyperedges/nets (squares and diamonds in the figure) and the nonzeros are its vertices (black circles).

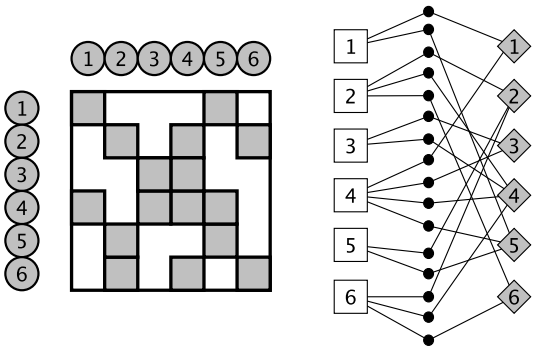



Fig. 1: A $6 \times 6$ sparse matrix $\mathbf{A}$ on the left and its corresponding hypergraph for the sparse-matrix partitioning problem; the nonzeros in the matrix are shown with grey squares. The white squares are empty.

Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets $\mathcal{N}$ among those vertices. A net $n \in \mathcal{N}$ is a subset of vertices and the vertices in $n$ are called its *pins*. Vertices can be associated with weights, denoted with $\mathsf{w}[\cdot]$, and nets can be associated with costs, denoted with $\mathsf{c}[\cdot]$. In this work, for the sparse-matrix bipartitioning problem, all the weights and the costs are equal to one.

A *$K$-way partition* of a hypergraph $\mathcal{H}$ is denoted as $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ where

- parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$,
- each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, i.e., $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$,
- union of $K$ parts is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$.

Let $W_k$ denote the total vertex weight in $\mathcal{V}_k$ (i.e., $W_k = \sum_{v \in \mathcal{V}_k} \mathsf{w}[v]$) and $W_{avg}$ denote the weight of each part when the total vertex weight is equally distributed (i.e., $W_{avg} = (\sum_{v \in \mathcal{V}} \mathsf{w}[v])/K$). If each part $\mathcal{V}_k \in \Pi$ satisfies the *balance criterion*

$$W_k \leq W_{avg}(1+\varepsilon), \quad \text{for} \quad k = 1, 2, \ldots, K \tag{1}$$

we say that $\Pi$ is *$\varepsilon$-balanced* where $\varepsilon$ represents the maximum allowed imbalance ratio.

For a $K$-way partition $\Pi$, a net that has at least one pin (vertex) in a part is said to *connect* that part. The number of parts connected by a net $n$, i.e., *connectivity*, is denoted as $\lambda_n$. A net $n$ is said to be *uncut* (*internal*) if it connects exactly one part (i.e., $\lambda_n = 1$), and *cut* (*external*), otherwise (i.e., $\lambda_n > 1$).

The set of external nets of a partition $\Pi$ is denoted as $\mathcal{N}_E$. To accurately model the total communication volume [2], the *connectivity-1* metric has been used;

$$\chi(\Pi) = \sum_{n \in \mathcal{N}} \mathsf{c}[n](\lambda_n - 1) \ . \tag{2}$$

In this metric, each cut net $n$ contributes $\mathsf{c}[n](\lambda_n - 1)$ to the cutsize. The hypergraph partitioning problem can be defined as the task of finding a balanced partition $\Pi$ with $K$ parts such that $\chi(\Pi)$ is minimized. As mentioned before, the problem is NP-hard [12].

An example bipartitioning for the corresponding hypergraph of the sparse matrix in Figure 1 is given in Figure 2. Since vertex weights are all one, the weight distribution is perfectly balanced. Since the net costs are all one, the connectivity-1 metric given in (2) is equal to two.

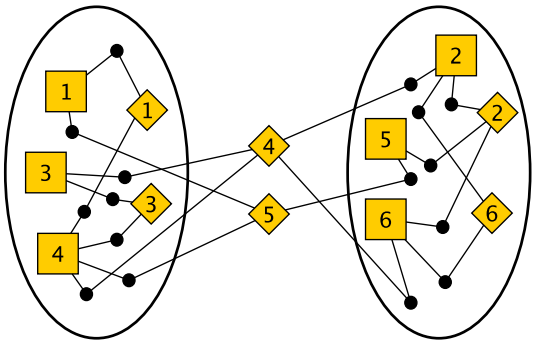



Fig. 2: A bipartitioning of the hypergraph in Figure 1. Part weights are equal (eight). The cut-size is two.

Figure 3 shows two different bipartitionings of the sparse matrix $\mathbf{A}$ in Figure 1. In the figure, the nonzeros assigned to the same part are colored with the same color. In the first bipartitioning, the red colored rows and columns generate cuts, i.e., they have two different colors among their nonzeros. Consider iterative SpMV, i.e., $\mathbf{Ax} = \mathbf{y}$; a red row implies communication since the corresponding result, a value in $\mathbf{y}$, will be partially computed in two processors and one of these partial results need to be transferred for the next iteration. Similarly, a red column implies communication since the corresponding $\mathbf{x}$ entry need to be transferred from one processor to another. Hence, although they are both perfectly balanced, the communication volume of the first bipartitioning

is eight and for the second one it is two. In fact, the second bipartitioning corresponds to the hypergraph partition in Figure 2.

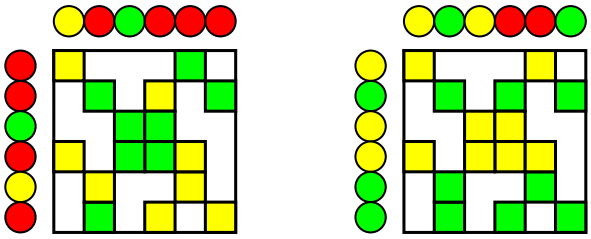


Fig. 3: Two different bipartitionings of the sparse-matrix $\mathbf{A}$ given in Figure 1. Eight and two entries, respectively, need to be transferred for each iteration of the corresponding SpMV.

*MondriaanOpt*: is a tool developed by Pelt and Bisseling [13]; it gets a sparse matrix and the allowed imbalance ratio $\varepsilon$ as inputs, and reports the optimal partitioning. The tool, which is released in 2016, is included in the Mondriaan package after Mondriaan v4.1 but it is independent and has a separate version number. Currently, the version number is v1.0.

*MondriaanOpt* uses a branch-and-bound-based algorithm and explores the search space in a depth-first manner over a tree; at each level a row or a column is assigned to part 0, part 1, or identified as a cut. Although, the algorithm prioritizes some rows/columns, e.g., the ones whose nonzeros are decided due to previous assignments, the initial ordering of the rows and columns matters. The tool uses multiple bounds for each node in the tree to derive a lower bound on the best communication volume. If this bound is more than the current best solution the algorithm stops going deeper and prunes the corresponding subtree rooted at the node being visited.

## III. Ordering techniques for optimal hypergraph partitioning

An ordering is basically a permutation of row and column ids which will be assigned during the branch-and-bound process. For all the orderings in this work, this permutation is determined at the beginning of the execution. Our main idea is assigning as much nonzeros as possible at the earlier levels of the search tree to make the bounds better approximate the real solution. We experimented on 3 different orderings as described below:

1) *Primary-Static (PS)* ordering simply sorts the rows and columns with respect to their number of nonzeros. There is no priority of the rows to columns or vice versa. Including the cost for counting the nonzeros, the complexity is $\mathcal{O}(m + n + nnz)$ where nnz is the total number of nonzeros in the sparse matrix.
2) *Primary-Dynamic (PD)* ordering takes the impact of a row/column assignment into account. That is, once a row/column is selected as the current one, it is removed from the matrix with all the nonzeros. Then the row/column with the maximum nonzero count is selected as the next one. For implementation, we generated a bucket for each nonzero count and placed the vertices to the buckets. Then starting with the bucket with the maximum id (nonzero count), we assigned the rows and columns. Throughout the process, the rows/columns are moved from one bucket to another. This happens for each nonzero in the matrix and the update cost is $\mathcal{O}(1)$. Hence, for this ordering, the overall complexity is $\mathcal{O}(m + n + nnz)$
3) *Secondary-Static (SS)* ordering concerns about total number of *secondary nonzeros* for a row/column; let's consider a row and its corresponding columns. We define the total number of nonzeros for all its columns as a secondary value for that row. The intuition is that once this row is assigned its columns are also touched and assigning a row that shares a nonzero with relatively denser columns is a logical decision to follow to enable the pruning at the earlier levels of the tree. For our implementation, the secondary values are computed in $\mathcal{O}(nnz)$ time. Overall, the complexity is $\mathcal{O}(m+n+nnz)$ with $\mathcal{O}(nnz)$ extra memory.

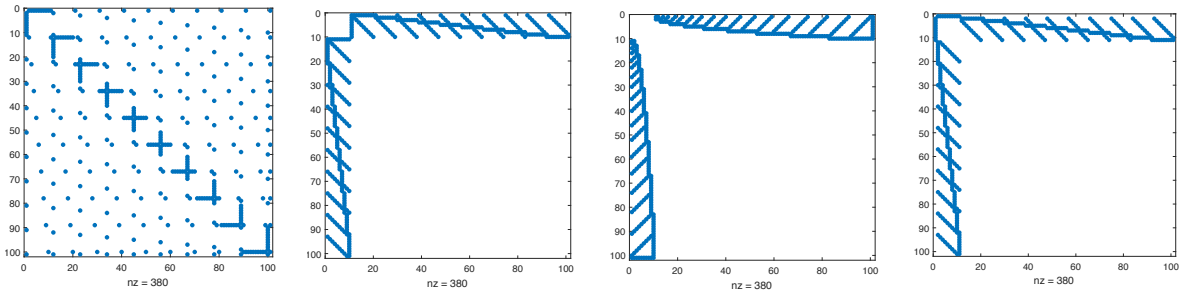
Figures 4 and 5 shows the impact of these orderings on the matrix structures for two matrices gd50 and Harvard500. Note that the figures only provide partial information, i.e., relative row/column orders. They do not show the actual permutation but clarifies our intention while choosing these orderings.

We used each ordering type in two different modes. Given a node (i.e., row/column), `MondriaanOpt` explores three options for that node; part 1, part 2, or cut. We modified `MondriaanOpt` to evaluate the impact of this order of these options. In our first mode, the cut option is explored **first** and the part options, part 1 and part 2, are explored later, respectively. The second mode explores the part options first and **lastly**, the cut option is explored. To distinguish these two modes we used the capital letters '**F**'irst and '**L**'ast. Overall we use six orderings which are named as PSL, PSF, PDL, PDF, SSL and SSF.

## IV. Choosing the best ordering based on the matrix features

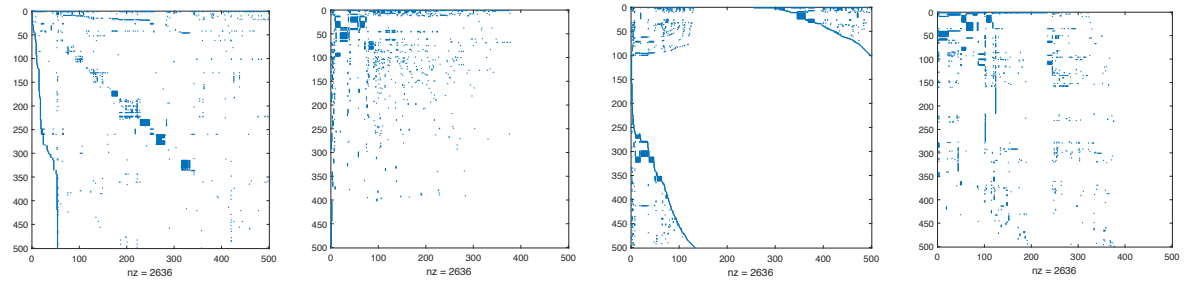
Although some of the orderings improve the execution time on average, we observed that they also can deteriorate the performance. Furthermore, their relative performance changes with respect to the matrix. We have experimented with many matrices and realized that it is hard to distinguish the matrix characteristics favoring a specific ordering. For this task, we generate a training dataset with the values obtained from the matrix features in Table I. In this dataset, a row corresponds to a matrix containing the 14 feature values and an additional value, *label*, which is the ID of the ordering with the fastest execution time; we also used natural ordering as a possible ordering to generate the training dataset.

We used Spark 1.6.1's machine learning library MLlib (`DecisionTree` and `DecisionTreeModel` classes from package `pyspark.mllib.tree`) to generate a decision tree from the training dataset. We then used a separate test dataset, which is similar to the training dataset but without




(a) Natural (b) PS: primary static (c) PD: primary dynamic (d) SS: secondary static

Fig. 4: Four different orderings for the gd06 matrix including the natural one (left-most), primary static (mid-left), primary dynamic (mid-right), and secondary static (right-most).




(a) Natural (b) PS: primary static (c) PD: primary dynamic (d) SS: secondary static

Fig. 5: Four different orderings for the Harvard500 matrix including the natural one (left-most), primary static (mid-left), primary dynamic (mid-right), and secondary static (right-most).

| | |
|---|---|
| 1 | Number of rows; $m$ |
| 2 | Number of columns; $n$ |
| 3 | Number of rows and columns; $m+n$ |
| 4 | Number of nonzeros; $nnz$ |
| 5 | Avg. nonzero count in a row; $nnz/m$ |
| 6 | Avg. nonzero count in a column; $nnz/n$ |
| 7 | Avg. nonzero count in a row/column; $nnz/(m+n)$ |
| 8 | Avg. bandwidth, i.e., dist. from the diagonal, w.r.t. the first diagonal |
| 9 | Avg. bandwidth, i.e., dist. from the diagonal, w.r.t. the second diagonal |
| 10 | PaToH volume / $(m+n)$ |
| 11 | PaToH volume / $\min(m,n)$ |
| 12 | Std. dev of nonzero counts of rows |
| 13 | Std. dev of nonzero counts of columns |
| 14 | Std. dev of nonzero counts |

TABLE I: Matrix features used to generate the decision tree.

the label column, to find the performance of the model. The details of the experiment is given in the next section.

## V. Experimental Results

All the simulation experiments in this section are performed on a single machine running on 64 bit CentOS 6.5 equipped with 384GB RAM and a dual-socket Intel Xeon E7-4870 v2 clocked at 2.30 GHz, where each socket has 15 cores (30 in total). Each core has a 32kB L1 and a 256kB L2 cache, and each socket has a 30MB L3 cache. All the codes are compiled with `gcc 4.8.4` with the `-O3` optimization flag enabled.

We used the matrices from `MondriaanOpt`'s webpage[2]. The matrices are classified as *minute*, *hour*, and *day*, where the class denotes the time spent by original tool to find the optimal bipartitioning. For all the experiments, we performed ten executions and reported the average result. Since the process can take days, we used upper limits on the maximum time; for hour matrices, the value is set to 7200 seconds, and for day matrices it is set to 216000 seconds. In the tables, when the optimal bipartitioning cannot be found within the maximum allowed time, the result is marked with $\infty$.

To use a good initial upper bound on the communication volume, before optimal bipartitioning, we bipartitioned the matrices into two with PaToH (with the quality option). We then feed `MondriaanOpt` with this initial value to prune the subtrees earlier on the search tree. Since the matrices are small, the initial bipartitioning time is not significant. However, the improvement on the execution time due to using a good initial bound is impressive; for instance, for the hour matrices, the optimal bipartitionings are found around 8.2× faster when PaToH is used to have an initial bipartitioning.

Tables II and III show the results of the first set of experiments for the hour and day matrices, respectively; the first column of each table shows the execution time in seconds for `MondriaanOpt` with the natural ordering. The next six columns show the normalized execution time (w.r.t. first column) of the optimal bipartitioning process with the orderings PSL, PSF, SSL, SSF, PDL, and PDF respectively. The last rows show the geometric mean of the normalized execution times; the one for natural ordering is set to one. As the tables

[2]http://www.staff.science.uu.nl/ bisse101/Mondriaan/Opt/

TABLE II: The execution time, in seconds, of `MondriaanOpt` with the natural ordering (first column) on the hour matrices, and the relative execution time of the tool with different orderings. The last row shows the geometric mean of all the normalized execution times for all orderings.

| Matrix | Natural | PSL | PSF | SSL | SSF | PDL | PDF |
|---|---|---|---|---|---|---|---|
| ash219 | 549.9 | 0.09 | 0.09 | ∞ | ∞ | 0.96 | 0.96 |
| ash85 | 488.7 | 0.51 | 0.51 | 1.23 | 1.24 | 0.43 | 0.43 |
| bcspwr03 | 266.3 | 0.73 | 0.08 | 2.02 | 0.11 | 0.44 | 0.07 |
| bcsstk01 | 4696.9 | 0.24 | 0.24 | 1.31 | 1.26 | 0.02 | 0.02 |
| Can_144 | 465.9 | 0.31 | 0.31 | 0.31 | 0.31 | 0.32 | 0.32 |
| ch5-5-b1 | 720.4 | 0.02 | 0.02 | ∞ | ∞ | 0.06 | 0.05 |
| Cities | 2121.4 | 0.43 | 0.74 | ∞ | ∞ | 0.02 | 0.00 |
| dolphins | 546.1 | 0.80 | 0.80 | 11.50 | 11.64 | 0.12 | 0.12 |
| Harvard500 | ∞ | ∞ | 0.13 | ∞ | ∞ | ∞ | 0.17 |
| impcol_b | 335.1 | 0.16 | 0.56 | 3.66 | 13.30 | 0.10 | 0.54 |
| lpi_klein1 | 213.3 | 2.00 | 1.81 | 9.70 | 14.42 | 0.90 | 0.20 |
| lp_kb2 | 173.4 | 3.03 | 2.41 | 17.12 | 5.53 | 0.02 | 0.02 |
| lp_recipe | 628.9 | 1.79 | 1.77 | 0.05 | 0.05 | ∞ | ∞ |
| lp_sc105 | 400.6 | 1.22 | 1.23 | 2.30 | 2.32 | 1.38 | 1.39 |
| lp_scagr25 | 404.3 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| lp_scsd1 | 504.6 | 0.02 | 0.02 | ∞ | ∞ | 0.02 | 0.02 |
| lp_stocfor1 | 131.2 | 8.04 | 44.85 | 4.88 | 38.24 | 1.24 | 7.35 |
| lp_vtp_base | 1106.6 | 0.38 | 0.39 | ∞ | ∞ | 0.25 | 0.36 |
| mesh1e1 | 303.7 | 0.42 | 0.42 | 1.72 | 2.19 | 0.25 | 0.25 |
| mesh1em1 | 303.9 | 0.42 | 0.42 | 1.80 | 1.73 | 0.25 | 0.25 |
| mesh1em6 | 339.5 | 0.38 | 0.37 | 1.54 | 1.54 | 0.22 | 0.22 |
| mhd3200b | 34.7 | 5.43 | 5.43 | 5.36 | 5.36 | 157.47 | 157.45 |
| mhd4800b | 105.2 | 10.78 | 5.83 | 5.74 | 5.74 | ∞ | ∞ |
| n3c5-b7 | 413.4 | 0.11 | 0.11 | 14.01 | 13.98 | 0.14 | 0.14 |
| nos4 | 2310.8 | 0.46 | 2.44 | 0.64 | 2.23 | 0.41 | 1.70 |
| odepa400 | 2.2 | 0.36 | 0.36 | 0.41 | 0.41 | 0.82 | 0.82 |
| polbooks | 401.3 | 0.77 | 0.78 | 2.70 | 2.72 | 0.16 | 0.16 |
| rajat11 | 903.8 | 0.08 | 0.74 | 6.63 | ∞ | 0.00 | 0.01 |
| steam3 | 62.2 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| west0067 | 73.8 | 0.14 | 0.14 | 0.47 | 0.47 | 0.48 | 0.51 |
| Wheel_4_1 | 327.4 | 0.25 | 0.25 | 1.16 | 1.17 | 0.23 | 0.23 |
| ww_36_pme | 51.5 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| geomean | 1.00 | 0.42 | 0.45 | 1.94 | 2.03 | 0.30 | 0.28 |

show, for the hour matrices, the PDF ordering reports 0.28 normalized execution time hence 3.6× improvement. For the day matrices, it is also the best one with 2.1× improvement over the natural ordering. For both matrix classes, the secondary value based orderings do not work.

A negative observation from the results is the orderings do not work for some matrices. Furthermore, for some classes such as *lp_* the proposed orderings make the bipartitioning problem unsolvable within the maximum allowed time. In addition, the relative performance of different orderings change with the matrices. We used the method described in Section IV to distinguish the matrix characteristics that favor a specific ordering. For training, we used three orderings; natural, PSL and PDF. For this experiment, we separated the dataset into two; the training set contains 79 matrices (minute, hour, and day) and similarly, the test data contains 19 matrices. The results of this experiment is given in Table IV. The last column in the table shows the normalized execution time of the predicted model. That is, given the matrix we ask the best ordering to the trained model and use that ordering for the execution. For this experiment, the best proposed ordering is PSF with 0.46 execution time. Column seven shows a hypothetical tool/model that always uses/chooses the best available ordering. The last

TABLE III: The execution time, in seconds, of `MondriaanOpt` with the natural ordering (first column) on the day matrices, and the relative execution time of the tool with different orderings. The last row shows the geometric mean of all the normalized execution times for all orderings.

| Matrix | Natural | PSL | PSF | SSL | SSF | PDL | PDF |
|---|---|---|---|---|---|---|---|
| add32 | 229.3 | 0.84 | 1.02 | 1.22 | 1.27 | 0.95 | 0.95 |
| ash331 | 33483.4 | 0.23 | 0.24 | ∞ | ∞ | 0.23 | 0.23 |
| bcsstk22 | 77794.5 | 0.52 | 0.86 | ∞ | ∞ | 0.16 | 0.16 |
| Bibd_9_5 | 5913.3 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| bwm2000 | 27342.2 | 0.15 | 0.19 | 0.13 | 0.13 | 0.24 | 0.19 |
| epb0 | 39974.5 | 3.70 | 3.48 | 3.49 | 3.03 | 3.33 | 3.32 |
| impcol_a | 187928.1 | 0.90 | ∞ | 0.28 | 0.28 | ∞ | ∞ |
| impcol_c | 71070.5 | 1.31 | 1.19 | 0.46 | 0.44 | 0.38 | 0.50 |
| laser | 35908.9 | 1.63 | 1.32 | ∞ | ∞ | 1.21 | 1.21 |
| Lns_131 | 88023.1 | 0.80 | 0.65 | 1.53 | 0.99 | 0.03 | 0.03 |
| lp_bore3d | 10074.1 | 2.79 | 1.99 | ∞ | ∞ | 0.12 | 0.12 |
| lp_share2b | 39460.9 | 0.94 | 0.76 | ∞ | ∞ | 0.30 | 0.30 |
| lp_ship08l | 4586.8 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| lp_ship08s | 7401.7 | ∞ | ∞ | 7.17 | ∞ | ∞ | ∞ |
| lp_ship12s | 85226.5 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| lung1 | 56207.5 | 0.20 | 0.16 | 0.59 | 0.59 | ∞ | ∞ |
| nos2 | 8267.6 | 0.14 | 0.28 | 0.22 | 0.26 | 0.25 | 0.28 |
| p0548 | 12201.4 | 1.07 | 1.15 | ∞ | ∞ | 0.97 | 0.89 |
| primagaz | 489.5 | 0.32 | 0.32 | 2.27 | 1.91 | 0.19 | 0.18 |
| tub1000 | 13840.9 | 0.30 | 0.22 | 0.21 | 0.35 | 0.24 | 0.20 |
| geomean | 1.00 | 0.69 | 0.69 | 1.33 | 1.42 | 0.48 | 0.47 |

TABLE IV: The results for the test dataset where the last two columns are for the best possible tool (with the proposed orderings) and the tool that uses the trained model.

| Matrix | Natural | PSL | PSF | PDL | PDF | Best | Predicted |
|---|---|---|---|---|---|---|---|
| add32 | 229.4 | 0.84 | 1.02 | 0.95 | 0.95 | 0.84 | 1.00 |
| bcspwr03 | 266.4 | 0.73 | 0.09 | 0.44 | 0.07 | 0.07 | 0.07 |
| bwm2000 | 27342.3 | 0.15 | 0.19 | 0.24 | 0.19 | 0.15 | 1.00 |
| ch5-5-b1 | 720.5 | 0.02 | 0.02 | 0.06 | 0.05 | 0.02 | 0.02 |
| cis-n4c6-b1 | 29.3 | 0.06 | 0.05 | 0.04 | 0.06 | 0.04 | 0.06 |
| epb0 | 39974.5 | 3.70 | 3.48 | 3.33 | 3.32 | 1.00 | 1.00 |
| gent113 | ∞ | 0.00 | 0.01 | 0.00 | 0.01 | 0.00 | 0.01 |
| Harvard500 | ∞ | ∞ | 0.13 | ∞ | 0.17 | 0.13 | ∞ |
| impcol_b | 335.0 | 0.16 | 0.57 | 0.10 | 0.54 | 0.10 | 0.54 |
| lp_recipe | 629.0 | 1.79 | 1.77 | ∞ | ∞ | 1.00 | 1.00 |
| lp_share1b | 33.8 | ∞ | ∞ | ∞ | ∞ | 1.00 | 1.00 |
| lp_ship08s | 0.2 | 60.71 | 81.81 | 259.69 | 276.53 | 1.00 | 1.00 |
| mhd3200b | 34.8 | 5.41 | 5.42 | 157.03 | 157.01 | 1.00 | 1.00 |
| n3c5-b7 | 413.4 | 0.11 | 0.11 | 0.14 | 0.14 | 0.11 | 0.14 |
| n4c6-b1 | 36.5 | 0.03 | 0.05 | 0.05 | 0.05 | 0.03 | 0.03 |
| primagaz | 489.6 | 0.32 | 0.32 | 0.19 | 0.18 | 0.18 | 1.00 |
| rajat11 | 903.9 | 0.08 | 0.74 | 0.00 | 0.01 | 0.00 | 0.01 |
| robot | 2.8 | 1.39 | 11.59 | 1.42 | 13.92 | 1.00 | 13.92 |
| tub1000 | 1.5 | 0.17 | 0.16 | 0.16 | 0.16 | 0.16 | 0.17 |
| geomean | 1.00 | 0.46 | 0.54 | 0.55 | 0.61 | 0.14 | 0.28 |

column shows the normalized execution times for the tool with the training. As the table shows, the training provided 1.6× improvement over PSF, 3.6× improvement over the natural ordering and only 2× worse than the best hypothetical tool. The decision tree for this generated by MLlib for this experiment is given in Algorithm 1.

## VI. Conclusion and Future Work

In this work, we use two ideas to make an optimal bipartitioning tool faster: 1) ordering the matrices to use more information in the earlier levels of the tree, and 2) designing a machine learning approach to choose the best ordering based

**Algorithm 1** The decision tree used in the experiments

```
if n ≤ 150.0 then
  if nnz/n ≤ 8.992684 then
    if nnz ≤ 160.0 then
      return natural
    else
      return PDF
  else
    if m ≤ 55.0 then
      return PDF
    else
      return PSL
else
  if Avg. bandwidth w.r.t. 1st diag ≤ 17.22028 then
    if m ≤ 1032.0 then
      return PSL
    else
      return natural
  else
    if nnz/m ≤ 24.4 then
      return natural
    else
      return PSL
```

on the matrix features. As our experiments on various matrices will show, these enhancements provides significant speedups over the original implementation.

In addition to the total data transfer, there are other communication metrics investigated before, e.g., total number of messages sent and/or received by a processor [16], [17]. The interplay of these metrics and their impact on performance of the sparse-matrix vector multiplication have been analyzed [11] and tools to simultaneously optimize more than one metric have been developed [18], [19]. These metrics are also interesting from the theoretical and practical point of view. As far as we know, we do not have the combinatorial methods to find optimal (bi)partitionings for these metrics.

## REFERENCES


[1] Ü. V. Çatalyürek and C. Aykanat, "A hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers," in *Proc. International Conference on High Performance Computing*, Dec. 1995.

[2] ——, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[3] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Computing*, vol. 26, pp. 1519–1534, November 2000.

[4] C. Walshaw, M. G. Everett, and M. Cross, "Parallel dynamic graph partitioning for adaptive unstructured meshes," *Journal of Parallel and Distributed Computing*, vol. 47, pp. 102–108, December 1997.

[5] D. M. Pelt and R. H. Bisseling, "A medium-grain method for fast 2d bipartitioning of sparse matrices," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014, pp. 529–539.

[6] B. Hendrickson, "Graph partitioning and parallel solvers: Has the emperor no clothes?" in *Lecture Notes in Computer Science*, 1998, pp. 218–225.

[7] Ü. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," in *Encyclopedia of Parallel Computing*, 2011.

[8] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005. [Online]. Available: https://doi.org/10.1137/S0036144502409019

[9] Ü. V. Çatalyürek, K. Kaya, and B. Uçar, "Integrated data placement and task assignment for scientific workflows in clouds," in *DIDC'11, Proceedings of the Fourth International Workshop on Data-intensive Distributed Computing, San Jose, CA, USA, June 8, 2011*, 2011, pp. 45–54. [Online]. Available: http://doi.acm.org/10.1145/1996014.1996022

[10] O. Küçüktunç, K. Kaya, E. Saule, and Ü. V. Çatalyürek, "Fast recommendation on bibliographic networks with sparse-matrix ordering and partitioning," *Social Netw. Analys. Mining*, vol. 3, no. 4, pp. 1097–1111, 2013. [Online]. Available: https://doi.org/10.1007/s13278-013-0106-z

[11] K. Kaya, B. Uçar, and U. V. Catalyurek, "On analysis of partitioning models and metrics in parallel sparse matrix-vector multiplication," in *Proc. of the 10th Int'l Conf. on Parallel Processing and Applied Mathematics (PPAM)*, 2013.

[12] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, U.K.: Wiley–Teubner, 1990.

[13] D. M. Pelt and R. H. Bisseling, "An exact algorithm for sparse matrix bipartitioning," *J. Parallel Distrib. Comput.*, vol. 85, no. C, pp. 79–90, Nov. 2015.

[14] ——, "A medium-grain method for fast 2d bipartitioning of sparse matrices," in *28th IEEE International Parallel and Distributed Processing Symposium*, May 2014.

[15] Ü. V. Çatalyürek and C. Aykanat, "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, Apr 2001.

[16] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, pp. 1837–1859, June 2004.

[17] R. H. Bisseling and W. Meesen, "Communication balancing in parallel sparse matrix-vector multiplication," *Electronic Transactions on Numerical Analysis*, vol. 21, pp. 47–65, 2005.

[18] Ü. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar, "UMPa: A multi-objective, multi-level partitioner for communication minimization," in *10th DIMACS Implementation Challenge Workshop: Graph Partitioning and Graph Clustering*, Feb 2012, pp. 53–66, published in Contemporary Mathematics, Vol. 588, Editors D.A. Bader, H. Meyerhenke, P. Sanders, D. Wagner, 2013.

[19] M. Deveci, K. Kaya, B. Uçar, and Ü. V. Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *Journal of Parallel and Distributed Computing*, vol. 77, pp. 69 – 83, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514002275