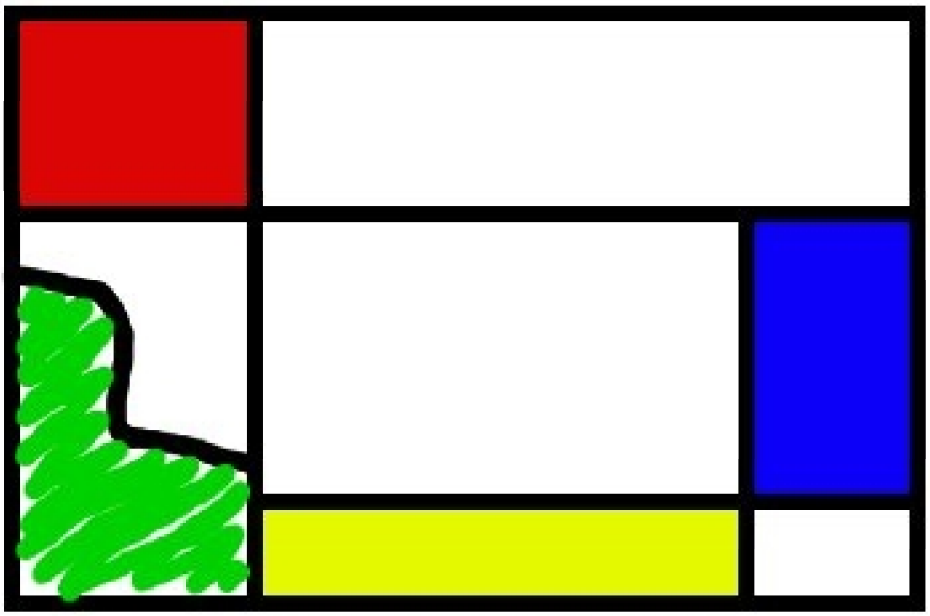
Expanding Mondriaan's Palette

a study to improve Mondriaan, a hypergraph-based matrix partitioner



Tristan van Leeuwen, supervisor: Rob Bisseling

Utrecht University

February 5, 2006

Preface

This thesis is the result of nine months work I have been doing to obtain my Master's degree in Computational Science at the University of Utrecht. Under the supervision of Dr. Rob Bisseling, I have been working on improving Mondriaan, a hypergraph-based matrix partitioner developed by him and Brendan Vastenhouw. After Brendan, another Master student, Wouter Meesen has worked on the vector partitioning of Mondriaan.

I would like to thank Rob Bisseling for having me working on Mondriaan, and for the free lunches; Ümit Çatalyürek, for helping me with my code and for the useful discussion on hypergraph partitioning; my roommates and fellow-students (Willem, Maartje, Marieke, Rob) for the regular cup of coffee and chat; and Willem in particular for helping me with the more sophisticated LaTeX tricks.

Contents

Chapter 1

Introduction

In science as well as in industry, the demand for computing power is growing. With problem size increasing and hardware price decreasing, parallel computers are becoming readily available. In the early days, it was tedious work to develop and optimize an algorithm for a specific parallel computer. It often happened that once the work was done, a faster computer was available and the not-so-portable algorithm had to be developed again for the new architecture. In the late 1980s the BSP (Bulk Synchronous Parallel) model was introduced by Valiant [32]. With this model and the matching library (BSPlib) it is possible to design and implement portable parallel algorithms with a predictable running time.

At the heart of many of these parallel algorithms lie sparse matrix-vector multiplications. To compute a matrix-vector product in parallel we will have to distribute the matrix and the vectors over the processors, and these processors will have to communicate. Since communicating takes time, it seems worth the effort to try to minimize the amount of communication. For this purpose, several graph-based models have been proposed. Good graph partitioners exist, but the graph models do not model the communication correctly. Hypergraph models overcome this problem, but the existing hypergraph partitioners are slower than the graph partitioners. In this thesis, we will focus on one such hypergraph-based matrix partitioner: Mondriaan [33], which was developed at the Utrecht University. We will try to make some improvements and incorporate another hypergraph based model in the existing software.

In Chapter one we will give a short, general overview of the graph and hypergraph models that have found widespread use. We will see that many other interesting problems from entirely different fields also give rise to a (hyper)graph partitioning problem. A few (hyper)graph partitioning algorithms are discussed, and finally we will elaborate on the Mondriaan package. In Chapter two, we present some new strategies for improving the current Mondriaan hypergraph partitioning algorithm and discuss experimental results. Chapter three

will present a way to incorporate another hypergraph model into Mondriaan, and results for this approach. In Chapter four we will elaborate on the hybrid method proposed in chapter three, and present results to compare the hybrid method with the original Mondriaan. In Chapter 5, the improvements made are tested on PageRank matrices. Finally, we draw conclusions and make recommendations for further development of Mondriaan.

1.1 Problem background: parallel matrix-vector multiplication

As said before, matrix-vector multiplications $\mathbf{u} = A\mathbf{v}$ lie at the heart of many computing applications. If we divide $\mathbf{u}, \mathbf{v}$, and A over P processors the parallel matrix-vector multiplication algorithm consists of four distinct phases:

1. **fan-out:** each processor gathers the vector elements it needs from the other processors,
2. **local:** each processor multiplies its local matrix elements with its local vector elements and sums them,
3. **fan-in:** each processor sends the local sums to the appropriate processors,
4. **final:** each processor adds the received sums.

To determine how the $m \times n$ matrix A and the vectors $\mathbf{u}, \mathbf{v}$ are distributed we will introduce mappings

$$\begin{aligned} \phi &: (i,j) \to \{0,\ldots,P-1\}, \\ \phi_{\mathbf{u}} &: i \to \{0,\ldots,P-1\}, \\ \phi_{\mathbf{v}} &: j \to \{0,\ldots,P-1\}, \\ &\text{for} \quad 0 \leq i < m, \quad 0 \leq j < n. \end{aligned} \tag{1.1}$$

Here, matrix element a_{ij} belongs to processor $\phi(i,j)$ and vector elements u_i, v_j belong to processor $\phi_{\mathbf{u}}(i)$, $\phi_{\mathbf{v}}(j)$ respectively. Please note that we let $\phi(i,j)$ map only non-zero elements. The parallel multiplication algorithm, adapted from algorithm 4.5 in [3] is listed in pseudo code in algorithm 1. In an actual implementation care must be taken to assure that vector elements are sent only once to each processor. Also, a processor should not have to sent to, or receive from itself.

Algorithm 1 Parallel Matrix-Vector Multiplication for processor s

Input: matrix A, distributed according to ϕ,

vector $\mathbf{v}$, distributed according to $\phi_{\mathbf{v}}$.

Output: vector $\mathbf{u} = A\mathbf{v}$, distributed according to $\phi_{\mathbf{u}}$.

{fan-out phase}

for all $(i,j) : j \in \phi_{\mathbf{v}}^{-1}(s)$ **do**

sent v_j to processor $\phi(i,j)$

end for

{local matrix-vector multiplication}

for all i **do**

$u_i^s = \sum_{j:(i,j)\in\phi^{-1}(s)} a_{ij} v_j$

end for

{fan-in phase}

for all $(i,j) : i \in \phi_{\mathbf{u}}^{-1}(s)$ **do**

get u_i^l from processor $l = \phi(i,j)$

end for

{final addition}

for all $i \in \phi_{\mathbf{u}}^{-1}(s)$ **do**

$u_i = \sum_{l=0}^{P-1} u_i^l$

end for

1.1.1 Generic matrix distributions

Now we can perform a matrix-vector multiplication with an arbritrary distribution. The next step is to find a good distribution. A good distribution will have the following properties:

- It will minimize the number of vector elements that are to be sent in the fan-out and fan-in phases (1, 3), and,
- It will distribute the work evenly, to minimize the time necessary for the local multiplication phase (2). We will refer to this as the load balance criterion.

We define the communication volume as the total number data words that is to be sent:

$$V = \sum_{0 \leq i < m, q_i \geq 1} (q_i - 1) + \sum_{0 \leq j < n, r_j \geq 1} (r_j - 1) \tag{1.2}$$

where

$$\begin{aligned} q_i &= |\phi(i, \{j\}_{0 \leq j < n})|, \\ r_j &= |\phi(\{i\}_{0 \leq i < m}, j)|, \end{aligned} \tag{1.3}$$

gives the number of processors owning row i and column j respectively. The vector elements u_i and v_j are assumed to belong to one of the processors in row i or column j respectively (i.e $\phi_{\mathbf{u}}(i) \in \phi(i, \{j\}_{0 \leq j < n})$ and $\phi_{\mathbf{v}}(j) \in \phi(\{i\}_{0 \leq i < m}, j)$). It is clear that minimizing the number of processors per row and column will minimize the communication volume. The load balance criterion reads:

$$\max_s |\phi^{-1}(s)| \leq (1 + \epsilon) \frac{|\phi^{-1}(\{s\}_{0 \leq s < P})|}{P}, \tag{1.4}$$

which simply states that all partitions are at most a fraction ϵ larger than the average partition size.

If the matrix A is full (which means that there are no non-zero elements), a block distribution suffices. The block distribution is defined as:

$$\phi(i, j) = \left\lfloor \frac{ir}{m} \right\rfloor + r \left\lfloor \frac{jq}{n} \right\rfloor. \tag{1.5}$$

Here, we divide the matrix into $P = qr$ blocks of equal size such that each row and column has only q and r processors owning it, respectively. The communication volume for a block distribution is

$$V_{\text{block}} = m(q - 1) + n(r - 1). \tag{1.6}$$

The optimal distribution is achieved for $q = \sqrt{\frac{n}{m} P}$, $r = \frac{P}{q}$. The block distribution satisfies the load balance criterion, and it minimizes the communication volume. A disadvantage is that this may lead to unbalanced communication. For methods to improve this, we refer to section 4.4 of [3].

When the matrix A is sparse, the situation is somewhat different. We denote the number of nonzero elements of a sparse matrix as $\mathrm{nz}(A)$. We could, of course, still use a block distribution, but this would most likely fail to satisfy the load balance criterion. It is not hard to assign each processor the same amount of work, but given this restriction, we will have to minimize the communication volume. One solution is to generalize the block distribution. We will call this the generalized block distribution. As with the block distribution, every row and column is divided in no more than q and r parts respectively. But these blocks do not have to have the same size. Thus, the communication volume for this distribution is less than or equal to that of the block distribution. Equality is reached when the blocks have no empty rows or columns. With the generalized block distribution we have more freedom to obtain a good load balance.

Another solution is to use a one-dimensional distribution where we assign whole columns to processors. This way every processor has to receive from $P-1$ other processors in the fan-in phase and the communication in the fan-out phase is zero. The communication volume for a 1D distribution is:

$$V_{\mathrm{1D}} \leq m(P-1). \tag{1.7}$$

The upper bound is reached if there are no empty rows in the blocks. While this upper bound is worse than that of the generalized block distribution, it will be easier to obtain a good load balance.

Mondriaan uses the 1D distribution recursively in row and column direction to obtain a 2D partitioning. This way it is still relatively easy to obtain a good load balance, and the number of processors per row and column is less than with a 1D distribution. Figure 1.1 illustrates four different distributions. Section 1.3 will elaborate on the Mondriaan distribution.

1.1.2 (Hyper)graph models for partitioning

How do we find a good 1D distribution? In the early days, the problem was formulated as a graph partitioning problem, which is known to be NP-hard [15]. This means that a polynomial time algorithm to solve the problem cannot be found (unless $P = NP$), so a heuristic method must be used to find a near-optimal solution.

For a square $n \times n$ matrix A the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ has $|\mathcal{V}| = n$ vertices and there is an edge between vertex j and j' if there is a non-zero $a_{jj'}$ or $a_{j'j}$. The weight of vertex j is the number of non-zeros in column j. An edge between vertices j and j' has weight two if both $a_{jj'}$ and $a_{j'j}$ are nonzero, else it has weight one. See figure 1.2 for an example. We can think of the partitioning of the graph as a permutation of the original matrix, where we shift the columns belonging to one partition to one side and permute the rows correspondingly. The total weight of the cut edges equals the number of off-block-diagonal elements. This is unfortunately

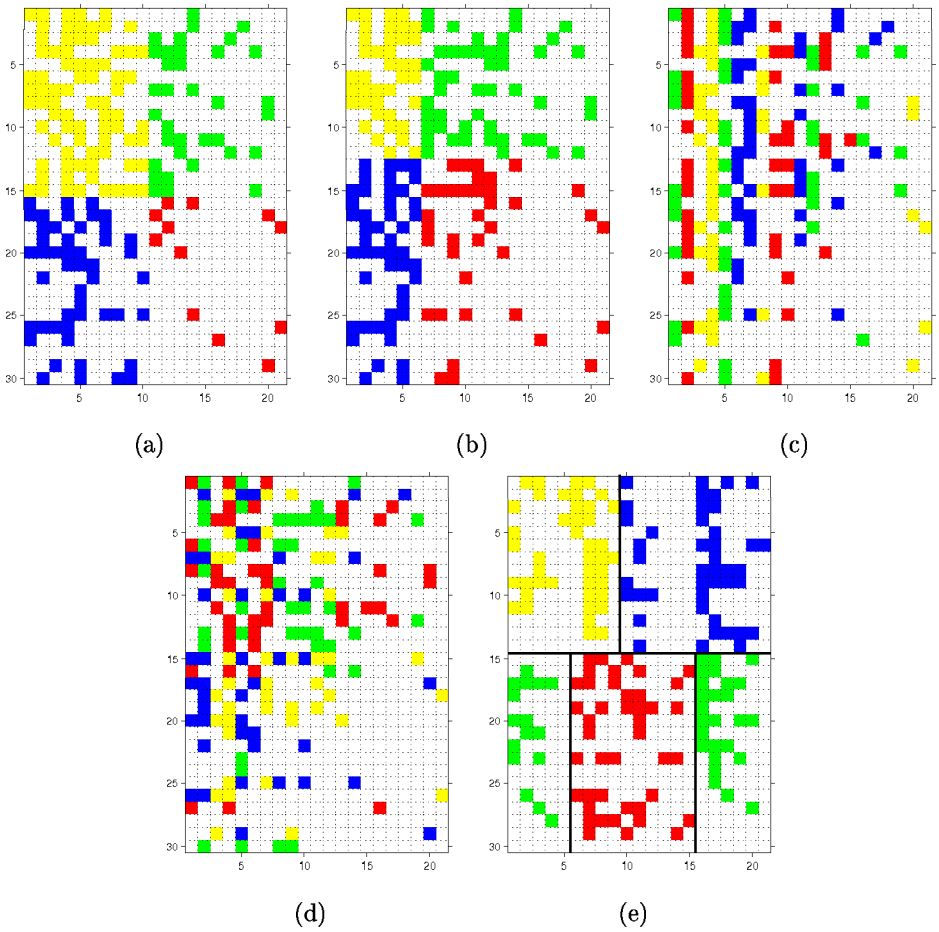


Figure 1.1: *Four different distributions over four processors. (a): Block distribution,* $V = 39$ *and* $\epsilon = 77\%$*. (b): Generalized block distribution,* $V = 43$*,* $\epsilon = 12\%$*. (c): 1D Distribution,* $V = 68$*,* $\epsilon = 3\%$*. (d): Mondriaan distribution,* $V = 40$*,* $\epsilon = 3\%$*. The first cut is done row-wise, the other two column-wise. (e): Permutation of (d); we can clearly see the resemblance to a painting of a certain painter.*

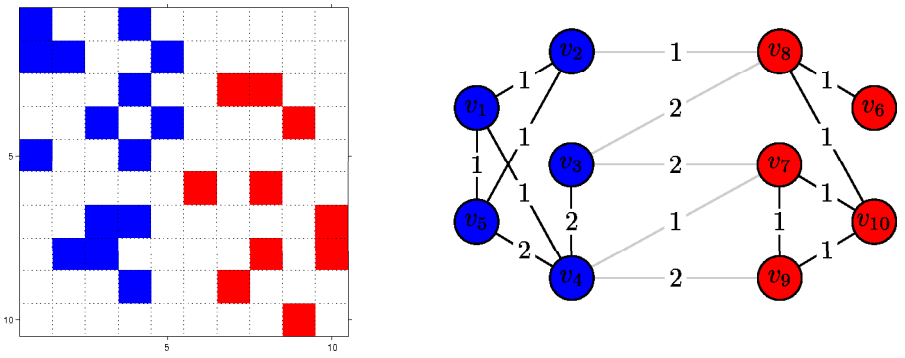


Figure 1.2: *Column-wise partitioned sparse matrix and its graph representation. The weights of the vertices are given by the number of non-zeros in the columns. The total weight of the cut edges is* 8*, while the actual communication volume is* 5*. Note that we could also use this graph to partition the matrix row-wise by changing the vertex weights.*

not exactly the communication volume we seek [19]. If there are for example two off-block diagonal elements in the same row, they will be sent as one partial sum. In this metric however, they are counted twice. To improve the metric, various ways for weighting the edges have been proposed [25, 18, 1, 26], which work especially well when the degrees of the vertices do not vary too much. Still, minimizing the number of off-block-diagonal non-zeros will help minimize the communication in the fan-in phase in some sense. We can also apply this approach to A^T for a row-wise partitioning of A. For rectangular matrices a similar graph can also be constructed, see [20]. To partition matrices this way, good graph partitioners have been developed, like Metis [23] and Chaco [21].

To model the communication accurately, a hypergraph model was introduced in [9]. A hypergraph $\mathcal{H}(\mathcal{V},\mathcal{N})$ is a generalization of a graph. In a hypergraph an edge (often called a net or hyperedge) can connect more than two vertices. Column j of the $m \times n$ matrix A represents a vertex $v_j \in \mathcal{V}$, and row i represents a net $n_i \in \mathcal{N}$. Net i connects all vertices j with $a_{ij} \neq 0$. For example, see figure 1.3. The hypergraph has n vertices and m nets. Now, each cut actually represents one data word that is to be sent, because communication arises when a row is divided. Minimizing the number of cuts will now minimize the true communication volume in the fan-in phase. We will refer to this approach as the row-net hypergraph model. Of course this approach can also be applied to A^T, which we will refer to as the column-net hypergraph model.

Another hypergraph approach has been proposed in [11]. In this hypergraph model, each non-zero element is viewed as a vertex. The nets are determined by the rows and columns. The fine-grained hypergraph of an $m \times n$ matrix A, having $\mathrm{nz}(A)$ nonzero elements is defined as follows. Each vertex is connected to

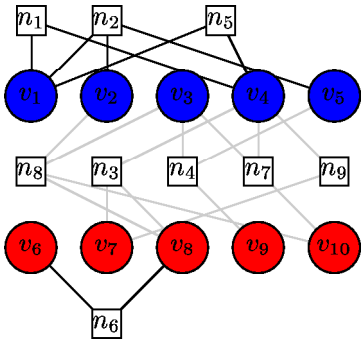


Figure 1.3: *Row-net hypergraph representation of the matrix from figure 1.2. The square nodes represent nets. We see that there are* 5 *broken nets, which is exactly the communication volume.*

all vertices in the same row by one net, and to all the vertices in the same column by another net. This hypergraph, having $m + n$ nets and $\text{nz}(A)$ vertices, is significantly larger that the row-net or column-net hypergraph. The communication in the fan-in and fan-out phases is minimized. This approach is referred to as the fine-grained hypergraph model. For more detail on the fine-grained hypergraph model we refer to Chapter 3.

1.2 To partition a (hyper)graph...

It turns out that (hyper)graph partitioning problems arise in many different fields. Close to home we have fill-reducing ordering [17, 34]. Here the objective is to permute a sparse matrix to block diagonal form in order to reduce fill-in when using direct methods on the matrix. (Hyper)graphs can be used in circuit design, where partitioning is used to determine the wiring schemes. They can be used to model some design process, where edges represent interdependencies between tasks [26]. In databases, relations between items can be modeled using a (hyper)graph. Partitioning will group those items that are relevant to each other together, which can be an aid in data mining [28]. In software engineering, the interdependencies in a software system can be represented by a call-graph. A good partitioning of the graph is wanted to divide the software system into manageable parts [4]. The graph partitioning problem also has some links to the graph coloring problem. With partitioning we want to assign each vertex a 'color' such that the number of adjacent vertices with different colors is minimized. In graph coloring we want the opposite, we want to maximize the number of adjacent vertices with different colors.

Over the past few decades, several methods for partitioning graphs have been proposed, and many of them have subsequently been adapted to partition hypergraphs. In this section, we will discuss a few of them:

Kernighan-Lin, Spectral partitioning, multilevel and recursive bisection.

1.2.1 Kernighan-Lin

This method was introduced in [24]. The basic idea is to find an initial partitioning and to improve this iteratively. The iterative improvement is done by moving vertices in an edge across the partition boundary. An efficient implementation was given in [12]. This method can be adapted to partition hypergraphs in two ways. The first is to construct a so-called clique-net graph of the hypergraph. In a clique-net graph all vertices in a given net have an edge between them. Various ways for weighting these edges have been proposed, see for example [25, 18, 1, 26]. The second is a direct generalization of the algorithm as in [9].

1.2.2 Spectral partitioning

With a (hyper)graph we can associate a Laplacian, which is defined below. In this method, the eigenvalues and eigenvectors of the Laplacian of the (hyper)graph are used to partition the matrix. See, for example [30]. The second eigenvalue of the Laplacian gives us information about the 'algebraic connectivity' of the graph and the second eigenvector defines a bi-partitioning of the graph into two connected subsets [13, 14]. We will briefly present some theory that may be useful to us.

For an undirected graph without loops or multiple edges, the Laplacian $\mathcal{L}$ is defined as:

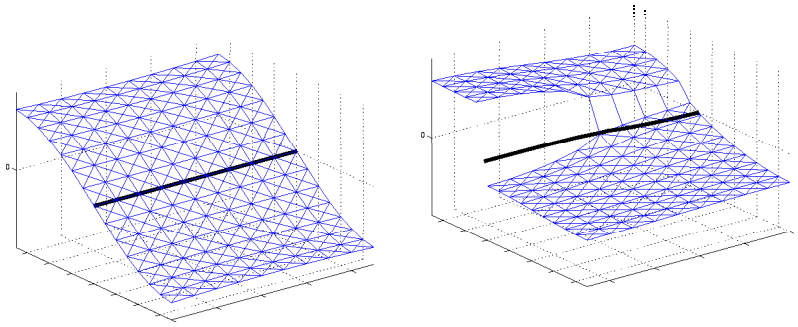
$$\mathcal{L} = D_v - A, \tag{1.8}$$

where D_v is the $n \times n$ diagonal matrix with the vertex degrees ν_i, and A is the $n \times n$ adjacency matrix of the graph. The matrix $\mathcal{L}$ is symmetric, so its eigenvalues are real and non-negative. We denote the eigenvalues as:

$$0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{n-1}. \tag{1.9}$$

It can be proven that $\lambda_0 = 0$ and that $\mathbf{w}_0 = (1, 1, \ldots, 1)^T$ is the corresponding eigenvector. Furthermore the multiplicity of the eigenvalue 0 gives the number of connected components of the graph. This means, for example, that if there are two eigenvalues 0 then the graph consists of two independent parts. From a matrix point of view this means that the adjacency matrix of this graph can be permuted to block-diagonal form, with no off-block-diagonal entries.

To partition the graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ into two partitions $\mathcal{G}_-(\mathcal{V}_-, \mathcal{E}_-)$, $\mathcal{G}_+(\mathcal{V}_+, \mathcal{E}_+)$, such that $\mathcal{V} = \mathcal{V}_- \cup \mathcal{V}_+$ and $\mathcal{E}_-, \mathcal{E}_+$ are the internal edges of the subgraphs, we can use the second eigenvector $\mathbf{w}_1$. We can associate each vertex i with a vector component $w_{1(i)}$. If we make a partition of all the vertices for which $w_{1(i)}$ is positive,



(a) (b)

Figure 1.4: *Two planer graphs, vibrating at their lowest eigenmode. The black lines are separators between the positive and negative parts. We see that the partitioning of graph (b) is indeed good in terms of number of cut edges, but it is not balanced.*

and a partition of the rest then the graphs $\mathcal{G}_-$ and $\mathcal{G}_-$ are connected (see Theorem 3.3 of [14]). We can interpret the second eigenvector as the lowest eigenmode of the graph. For a planar graph (i.e., a graph that can be represented on a plane without crossing edges) we can visualize this as the vibration of a plane. The idea is that the plane will vibrate around the 'weakest' point. See figure 1.4 for an example.

We can also use the Laplacian to derive a lower bound for the number of cut edges in terms of the second eigenvalue. Consider a vector $\tilde{\mathbf{w}}$ such that $\tilde{w}_i \in \{-1, +1\}$, $\sum \tilde{w}_i = 0$ (i.e. $\tilde{\mathbf{w}} \perp \mathbf{w}_0$). For brevity we will refer to the space spanned by such vectors as $\tilde{W}$. Now we can calculate

$$\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}} = \sum_{i,j} \mathcal{L}_{ij} \tilde{w}_i \tilde{w}_j =$$

$$\sum_{i=j} \nu_i + \sum_{i,j \in \mathcal{V}_-} \mathcal{L}_{ij} + \sum_{i,j \in \mathcal{V}_+} \mathcal{L}_{ij} - \sum_{i \in \mathcal{V}_-, j \in \mathcal{V}_+} \mathcal{L}_{i,j} - \sum_{i \in \mathcal{V}_+, j \in \mathcal{V}_-} \mathcal{L}_{i,j} = \tag{1.10}$$

$$2|\mathcal{E}| - 2|\mathcal{E}_-| - 2|\mathcal{E}_+| + 2|\mathcal{E} \setminus (\mathcal{E}_- \cup \mathcal{E}_+)| = 4|\mathcal{E}_{\text{cut}}|,$$

where $\mathcal{E}_{\text{cut}} = \mathcal{E} \setminus (\mathcal{E}_- \cup \mathcal{E}_+)$ is the set of edges between $\mathcal{V}_-$ and $\mathcal{V}_+$, and $|\mathcal{E}_{\text{cut}}|$ gives us the number of cut edges. To minimize the number of cut edges we simply seek a $\tilde{\mathbf{w}}$ which minimizes (1.10). Now we recall that we can calculate $\lambda_1, \mathbf{w}_1$ by minimizing the Rayleigh quotient (see for example [16]):

$$\lambda_1 = \min_{\mathbf{w} \perp \mathbf{w}_0} \frac{\mathbf{w}^T \mathcal{L} \mathbf{w}}{\|\mathbf{w}\|_2^2}. \tag{1.11}$$

We can relate (1.10) and (1.11) as:

$$\min_{\mathbf{w} \perp \mathbf{w}_0} \frac{\mathbf{w}^T \mathcal{L} \mathbf{w}}{\|\mathbf{w}\|_2^2} \leq \min_{\tilde{\mathbf{w}} \in \tilde{W}} \frac{\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}}}{|\mathcal{V}|}, \tag{1.12}$$

because the space spanned by $\mathbf{w} \perp \mathbf{w}_0$ is larger than $\tilde{W}$. Therefore:

$$|\mathcal{E}_{\text{cut}}| \geq \frac{\lambda_1 n}{4}, \tag{1.13}$$

where $n = |\mathcal{V}|$, gives a lower bound on the number of cut edges for a perfectly balanced bi-partition of the graph $\mathcal{G}$. Note that n must be even.

The theory can also be expanded to include hypergraphs. The Laplacian of a hypergraph is defined as [5]:

$$\mathcal{L} = D_v - A^T D_e^{-1} A. \tag{1.14}$$

where D_v is an $n \times n$ diagonal matrix with the vertex degrees ν_j, D_e is an $m \times m$ diagonal matrix with the edge sizes α_i, and A is the $m \times n$ adjacency matrix corresponding to the row-net hypergraph. Note that in the case of a graph, this definition is the same as (1.8), except for a factor 2. The Laplacian of a hypergraph is square and symmetric. Also, $\lambda_0 = 0$ and $\mathbf{w}_0 = (1, 1, \ldots, 1)$ is the corresponding eigenvector. We define a bi-partitioning by $\tilde{\mathbf{w}}$ in the same way as before. To derive a lower bound on the number of cut nets, we calculate

$$\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}} = \sum_{i,j} \mathcal{L}_{ij} \tilde{w}_i \tilde{w}_j = \sum_i \tilde{w}_i^2 \nu_i - \sum_{i \neq j} \tilde{w}_i \tilde{w}_j \sum_l \frac{a_{li} a_{lj}}{\alpha_l}. \tag{1.15}$$

Using the fact that $\mathcal{L}\mathbf{w}_0 = 0$, we can write:

$$\sum_i \nu_i = \sum_{i,j} \sum_l \frac{a_{li} a_{lj}}{\alpha_l}.$$

Since $\tilde{w}_i^2 = 1$, equation (1.15) now becomes:

$$\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}} = \sum_{i,j} (1 - \tilde{w}_i \tilde{w}_j) \sum_l \frac{a_{li} a_{lj}}{\alpha_l}. \tag{1.16}$$

As before we can split the sum, and the terms where $\tilde{w}_i \tilde{w}_j = 1$ will cancel out, leaving:

$$\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}} = 2 \sum_{i \in \mathcal{V}_-, j \in \mathcal{V}_+} \sum_l \frac{a_{li} a_{lj}}{\alpha_l} + 2 \sum_{i \in \mathcal{V}_+, j \in \mathcal{V}_-} \sum_l \frac{a_{li} a_{lj}}{\alpha_l}. \tag{1.17}$$

Unfortunately this will not give us the number of cut nets, as with graphs. However, we observe that:

$$\sum_{i \in \mathcal{V}_+, j \in \mathcal{V}_-} \sum_l \frac{a_{li} a_{lj}}{\alpha_l} = \sum_{i \in \mathcal{V}_-, j \in \mathcal{V}_+} \sum_l \frac{a_{li} a_{lj}}{\alpha_l} = \sum_{\text{cut nets } l} \frac{(\alpha_l - \alpha_l^+)\alpha_l^+}{\alpha_l}, \tag{1.18}$$

where α_l^+ gives the number of vertices net l has in partition $\mathcal{V}_+$ and $\alpha_l - \alpha_l^+$ gives the number of vertices net l has in partition $\mathcal{V}_-$. We define the weighted communication volume as:

$$\tilde{V} = \sum_{\text{cut nets } l} \frac{(\alpha_l - \alpha_l^+)\alpha_l^+}{\alpha_l}. \tag{1.19}$$

This weighted volume does not only measure the number of cut nets, but also takes into account the imbalance in the cut. For our purpose this is not immediately useful information. Following the same arguments as with graphs we can write:

$$\tilde{V} \geq \frac{\lambda_1 n}{4}. \tag{1.20}$$

We can also give a lower bound for the communication volume V (which equals the number of cut nets in the case of a bi-partitioning) in terms of the weighted volume by observing that the weighted volume has a maximum when all cut nets have an equal number of vertices in both partitions (i.e. $\alpha_l^+ = \frac{1}{2}\alpha_l$):

$$\tilde{V} \leq \frac{1}{4} \sum_{\text{cut nets } l} \alpha_l \leq \frac{1}{4} V \max_l \alpha_l. \tag{1.21}$$

We now have a lower bound for the (weighted) communication volume of a balanced bi-partitioning. An important note to this lower bound is that the partition defined by the second eigenvector is not necessarily a balanced partition. Moreover, using the eigenvector for partitioning minimizes the weighted communication volume, which is the wrong metric for our purpose. Still, this theoretical insight can be put to use as we will see later.

1.2.3 Multilevel partitioning

This technique was first proposed in [8], and is especially useful when the (hyper)graph in question is very large. By merging 'similar' vertices in several iterations the size of the graph is diminished; this process is referred to as 'coarsening'. The coarsened graph can then be partitioned using any method. Finally, the graph is un-coarsened by taking the vertices apart, and refined by trying to swap certain vertices. Coarsening the graph will also give some global information about the graph. When using KL to do the initial partitioning this is particularly beneficial since KL is known to get stuck in local minima. The Mondriaan package also uses this technique, and Chapter 2 will elaborate on this subject.

1.2.4 Recursive bisection

As opposed to k-way partitioning, where we divide the matrix into k partitions at once, there is recursive bisection. With this technique we repeatedly divide the matrix in two, and after $\log_2 k$ steps we have k partitions. This method can also be made to work when k is not a power of two. The advantage is that we can, for example, alternately divide the matrix row wise and column wise. This 2D partitioning is a key feature of the Mondriaan package. We refer to section 1.3.1 for more details.

1.3 The Mondriaan partitioning software

The Mondriaan package [33] combines several of the above mentioned techniques to find a partitioning $\phi(i,j)$, $\phi_{\mathbf{u}}(i)$, $\phi_{\mathbf{v}}(j)$ such that the communication volume V_ϕ is minimized and that the load balance criterion (1.2) is satisfied. Please note that we assume that ϕ maps only non-zero elements.

For convenience we will now view the matrix A as an index set of non-zero elements:

$$A = \{(i,j) : 0 \leq i < m \wedge 0 \leq j < n \ \wedge a_{ij} \neq 0\}, \quad \text{for} \quad 0 \leq s < P. \tag{1.22}$$

The partitions A_s can be viewed as disjoint subsets of A such that $A = \bigcup_{0 \leq s < P} A_s$. The partitions A_s are determined by ϕ as:

$$A_s = \{(i,j) : \phi(i,j) = s\}. \tag{1.23}$$

The size of the partition can now be written as $|A_s|$. With $V(A_0, \ldots, A_{P-1}) = V_\phi$ we denote the communication volume induced by the partitioning of A into $A_0, \ldots, A_{P-1}$.

In Mondriaan, a multilevel recursive bi-partitioning algorithm is applied. The general algorithm of Mondriaan is given in algorithm 2. Essentially Mondriaan does a recursive bi-partitioning of the matrix, where each recursion step the optimal direction (row or column) is chosen by trying both. A key feature of Mondriaan is the way the load balance is maintained. The allowed imbalance for each bi-partitioning is dynamically adjusted along the way. How the bi-partitioning is done is discussed in the next section.

1.3.1 The bi-partitioning algorithm

At the heart of Mondriaan lies the bi-partitioning algorithm, which tries to split a matrix A in two parts A_0, A_1 while minimizing the communication volume $V(A_0, A_1)$ and maintaining the load balance. An important property of this bi-partitioning technique is that the partitions can be split independently. Take, for example, a partitioning into 4 parts. To minimize the communication volume, we can minimize the volume of the first bi-partitioning and the subsequent two bi-partitionings independently, since:

$$V(A_0, A_1, A_2, A_3) = V(A_0 \cup A_1, A_2 \cup A_3) + V(A_0, A_1) + V(A_2, A_3). \tag{1.24}$$

For a proof we refer to [33]. Of course one could use any of the partitioning techniques mentioned in the previous section for this. The bi-partitioning algorithm in Mondriaan however, uses multilevel Kernighan-Lin.

As explained before we can interpret the matrix A as the adjacency matrix of a hypergraph. In Mondriaan the row-net, and column-net hypergraphs are used. In the following description we will assume the row-

Algorithm 2 MatrixPartition(A,P,ϵ)

Input: sparse matrix A, number of partitions P and allowed imbalance ϵ.

Output: partitions $A_0, \ldots, A_{P-1}$ such that $\max_s |A_s| \leq (1+\epsilon)\frac{|A|}{P}$.

if $P > 1$ **then**

$maxnz = (1+\epsilon)\frac{|A|}{P}$

$q = \log_2 P$

(A_0^{ROW}, A_1^{ROW})=BiPartition(A,ROW,ϵ/q)

(A_0^{COL}, A_1^{COL})=BiPartition(A,COL,ϵ/q)

if $V(A_0^{ROW}, A_1^{ROW}) > V(A_0^{COL}, A_1^{COL})$ **then**

$(A_0, A_1) = (A_0^{COL}, A_1^{COL})$

else

$(A_0, A_1) = (A_0^{ROW}, A_1^{ROW})$

end if

$\epsilon_0 = \frac{maxnz}{|A_0|}\frac{P}{2} - 1$

$\epsilon_1 = \frac{maxnz}{|A_1|}\frac{P}{2} - 1$

MatrixPartition(A_0,$P/2$,ϵ_0)

MatrixPartition(A_1,$P/2$,ϵ_1)

else

return A

end if

net hypergraph, but of course it is equally applicable to the column-net hypergraph. The multilevel KL algorithm consists of three distinct phases:

Coarsening phase: In this phase, the size of the hypergraph is diminished to speed up the computation. This is done by pairwise merging of vertices that are 'similar'. The pairwise merging halves the number of vertices each iteration, and it is repeated until the hypergraph is sufficiently small. Mondriaan uses the HCM (Heavy Connectivity Matching) strategy to merge the vertices. This measures how many nets two vertices have in common and merges two vertices with the highest overlap. From a matrix point of view this means that we simply calculate the inner product between columns of the matrix and merge the two vertices with the largest inner product. Of course this is just one of the many possible similarity measures, and this is discussed in Chapter 2.

Initial bi-partitioning: When the hypergraph is small enough, a random partitioning is done and it is improved using the Fiduccia-Matthyses [12] implementation of the KL algorithm. The process is repeated several times using different initial random partitionings and the best is taken. Throughout this phase the load-balance is maintained.

Un-coarsening/refinement phase: Now it is time to take the merged vertices apart, which is referred to as un-coarsening. This is done in several levels, doubling the number of vertices at each level. After each level several KL passes are performed to try to improve the partitioning. This is referred to as refining.

1.3.2 Vector partitioning

For parallel matrix-vector multiplication $\mathbf{u} = A\mathbf{v}$, we also need to partition the vectors $\mathbf{u}$ and $\mathbf{v}$. Basically, a vector element u_i is assigned to any of the processors owning matrix row i and a vector element v_j is assigned to any of the processors owning matrix column j. Randomly picking any of the processors in a row or column may result in an unbalanced distribution of the vectors. Recently, this problem has been addressed in [2].

In some special cases we would like both vectors to have the same distribution. This happens when we want to add or multiply the vectors with each other, or in a repeated matrix-vector multiplication (e.g. the power method). To accommodate this, the vectors are partitioned according to the matrix diagonal. When needed, dummy diagonal elements are added to the matrix. This can induce extra communication, since it may happen that a vector element is assigned to a processor that is not owning a non-zero in the corresponding row and/or column.

Chapter 2

Improvements of the coarsening phase

In this chapter we will discuss a few variants of the inner product measure that can be used to merge vertices in the coarsening phase. The goal is to find one that works better than the one currently used in Mondriaan. We do not strive for completeness but we try a few variants that are intuitively appealing, and we try to explain why they do, or do not work. Apart from scaling the inner product, we will also try different matching orders. First, we will give a brief overview of the coarsening algorithm. Several ways of scaling the inner product will be discussed and we will present a graphical way of comparing them, based on [22]. This will give us some idea of how the different scalings behave. Some theoretical justification for the scaling will also be sought. Subsequently, we will discuss experimental results.

2.1 Inner product matching

We will discuss the algorithm from a matrix point of view. Recall that the columns of the matrix A define the vertices, and that the rows define the nets, where each net i connects all vertices j for which $a_{ij} \neq 0$. We will denote the degree of vertex j by ν_j, and the size of net i by α_i. Each vertex has a weight equal to the number of non-zeros in the corresponding column. The weight is used for load balancing. To measure the similarity of matrix columns we essentially use the inner product, although other measures have also been proposed (see for example [28, 6]).

So, to merge the vertices we essentially have to calculate A^TA and merge the vertices that have the highest inner product. But, since it is quite expensive to calculate A^TA, we would like to speed up the computation. To achieve this Mondriaan uses a greedy algorithm. The vertices are visited in a certain order, and then the inner product with the current vertex, or candidate, and all other unmatched vertices is calculated. The vertex that gave the highest inner product is merged with the candidate. A vertex that has been merged

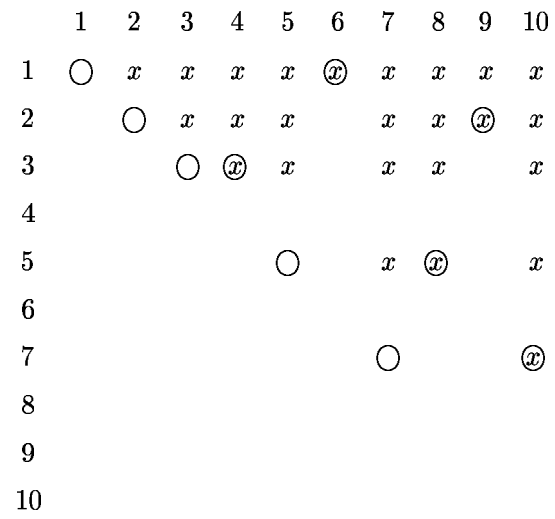


Figure 2.1: *Example of the A^TA matrix for a matrix A with 10 vertices that are all connected (i.e. A^TA is a full matrix). The visiting order is the natural order of the rows. The elements denoted by x are calculated entries, the encircled entries indicate a match between the corresponding vertices. Each time a match is made, a row and a column are eliminated from the computation. Using the greedy algorithm cuts the computation cost from $\frac{1}{2}n(n-1)$ inner product calculations to $\frac{1}{4}n(n-4)$ inner product calculations when A^TA is a full matrix. In the case that A^TA is not a full matrix, zero entries are not calculated and the gain may be less.*

will not be visited again, thus cutting the computation time. See figure 2.1 for an example. We will refer to a pair of merged vertices as a cluster. The weight of a cluster is the sum of the weights of the vertices in it, i.e., the sum of the number of non-zeros in the columns. Once all vertices have have been merged we have a hypergraph whose number of vertices is roughly half that of the original hypergraph. We can apply the procedure again on the coarsened hypergraph, until the number of vertices is sufficiently small. The coarsening algorithm is presented in algorithm 3.

Algorithm 3 Coarsen Hypergraph

Input: Hypergraph $\mathcal{H}_0(\mathcal{V}_0, \mathcal{N}_0)$ with $|\mathcal{V}_0| = n$ vertices and $|\mathcal{N}_0| = m$ nets

Output: Coarsened Hypergraph $\mathcal{H}_k(\mathcal{V}_k, \mathcal{N}_k)$, with $|\mathcal{V}_k|$ <AllowedNrVertices vertices and $|\mathcal{N}_k| = m$ nets

```
k = 0
w_total = Σ_i w_i
while |V_k| > AllowedNrVertices do
  while V_k ≠ ∅ do
    pick candidate v_i from V_k according to matching order
    V_k = V_k \ v_i
    for all j ∈ V_k do
      IP_j = M(v_i, v_j)
    end for
    find j_max s.t IP_jmax ≥ IP_j   ∀j and w_i + w_jmax ≤ 0.2 w_total
    if IP_jmax ≠ 0 then
      v = merge(v_i, v_jmax)
      w_v = w_i + w_jmax
      V_{k+1} = V_{k+1} ∪ v
      V_k = V_k \ v_jmax
    end if
  end while
  k = k + 1
end while
```

We will restrict ourselves to varying the visiting order of the candidates, and scaling the inner product. Other parameters, such as the number of vertices to merge per pass (2 vertices), the allowed weight of a cluster (20% of the total weight) and the preferred size of the coarsened hypergraph (200 vertices), are taken to be Mondriaan default values (stated in brackets). The visiting order is simply one of 'random', 'increasing weight', 'decreasing weight' or 'natural'. With the increasing weight order, we will first find a match for the heaviest vertex. Natural order is the order as determined by the matrix. We will discuss the various scalings

$$
\left.\begin{array}{ccccc} 0 & 0 & 0 & \hat{\mathbf{1}} & \mathbf{0} \\ 1 & 1 & 0 & \mathbf{0} & \tilde{\mathbf{1}} \\ 1 & 0 & 0 & \hat{\mathbf{1}} & \mathbf{0} \end{array}\right\} \text{possible external connections}
$$

$$
\left.\begin{array}{ccccc} 0 & 1 & 1 & \mathbf{1} & \mathbf{1} \\ 1 & 1 & 0 & \mathbf{1} & \mathbf{1} \end{array}\right\} \text{internal connections}
$$

Figure 2.2: *The boldfaced columns are in the same cluster. The possible external connections are extra connections the cluster might have w.r.t its vertices. The HCM measure does not measure these external connections. The cosine measure measures all possible external connections, the min measure measures only the* $\tilde{1}$ connection and the max measure measures the $\hat{1}$ connections.

we have applied to the inner product below.

2.1.1 Column Scaling

Here we will consider a column-wise scaling. The idea of this scaling is to measure not only connections in a cluster, but also the possible external connections. It is clear that we would not like a cluster with more connections outside the cluster than inside it. See figure 2.2 for a schematic depiction of the situation.

The general formula for the column-scaled inner product between column j and j' is:

$$M(j,j') = \frac{1}{\omega(j,j')} \sum_{i=0}^{m-1} a_{ij} a_{ij'}. \tag{2.1}$$

Note that we interpret A as the adjacency matrix, so its elements are either one or zero. The different choices of $\omega(j,j')$ are discussed below.

HCM

Here we simply take the inner product between two matrix columns, thus measuring the overlap. It is clear that this overlap should be as high as possible to ensure that the number of connections in the cluster is maximized. The number of connections outside the cluster is not taken into account. The weighting factor is thus simply:

$$\omega(j,j') = 1. \tag{2.2}$$

A disadvantage of the HCM measure is that a perfect match cannot be discerned. The maximum value depends on the smallest degree, but as seen in figure 2.3, this maximum is not unique.

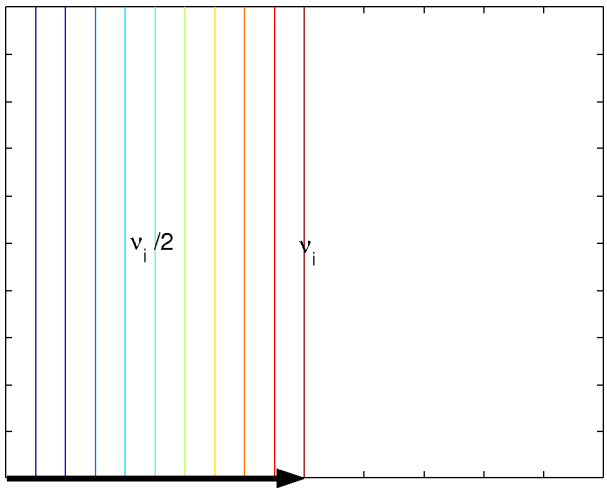


Figure 2.3: *Shown is the plane spanned by the two vectors that are to be matched, and the contour plot of the HCM measure applied to the drawn vector and any other vector in the plane. The length of a vector gives the number of non-zeros in the corresponding columns. The overlap is given by the projection on the x axis. A difference in angle and/or length means that there are possible external connections. Red is the maximum, which depends on the the degree ν_i of the candidate vertex. This measure does not distinguish between matches with the same overlap and a different number of external connections.*

Cosine

In the HCM measure we do not take the degrees of the vertices in account. To see why this could be desirable we look at an example. Let $a_{*j} = (1,1,0,1,1,0)^T$, $a_{*j'} = (1,1,0,0,0,0)^T$ and $a_{*j''} = (1,1,1,0,0,1)^T$. Now $M_{\text{HCM}}(j,j') = M_{\text{HCM}}(j,j'') = 2$, but a cluster of j,j'' may have external connections, whereas a cluster of j,j' does not. This motivates us to take the degree of the vertices into account. One way of doing this is by normalizing the inner product as:

$$\omega_{\text{cosine}}(j,j') = \sqrt{\nu_j \nu_{j'}}. \tag{2.3}$$

This measure reflects the external connections that j,j'' may have: $M_{\text{cosine}}(j,j') = \frac{1}{\sqrt{2}} > M_{\text{cosine}}(j,j'') = \frac{1}{2}$. This measure also gives the cosine of the angle the two vectors make, hence the name. Another advantage is that we have an upper bound for the measure, where the maximum is reached for a perfect match (i.e. the columns are identical). In figure 2.4 a contour-plot for this measure is depicted. This measure is widely used in information retrieval, where the columns represent documents and the non-zero elements are terms. For more on the use of similarity measures in information retrieval, see [27].

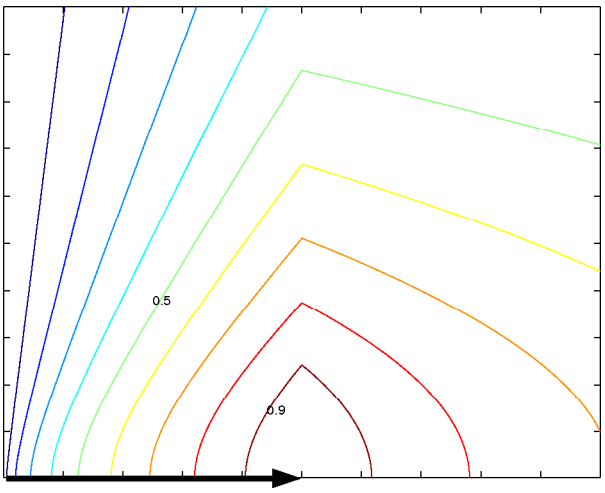


Figure 2.4: *Contour plot of the cosine measure. Here, the maximum value of the measure is one. This maximum is reached only if the vectors are identical. Every possible external connection results in a lower value.*

Min

A variation on the Cosine measure is the Min measure. This measure prefers clusters where column j is a subset of column j', thus generating no external connections. Please note that this measure minimizes the number of possible external connections w.r.t the vertex with the largest degree. Possible external connections w.r.t the vertex with the smallest degree are not measured. The weighing factor is:

$$\omega_{\min}(j, j') = \min\{\nu_j, \nu_{j'}\}. \tag{2.4}$$

Max

Now it seems natural to also define the Max measure. This measures possible external connections w.r.t the smallest vertex. The maximum value is one, which is achieved only if the vertices are identical, as with the cosine measure. The main difference with the cosine measure is the way the possible external connections are measured. See figure 2.6.

$$\omega_{\max}(j, j') = \max\{\nu_j, \nu_{j'}\}. \tag{2.5}$$

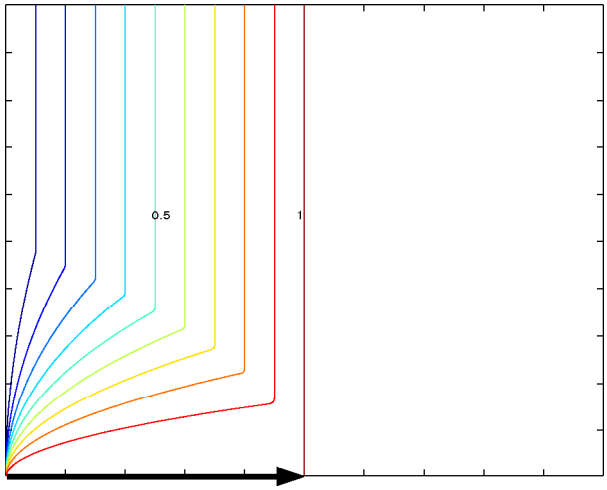


Figure 2.5: *Contour plot of the min measure. The maximum value is one, which is achieved for any vertex that is a subset of the largest vertex (i.e there are no extra connections w.r.t the largest vertex). Here, only external connections w.r.t the largest vertex result in a lower value.*

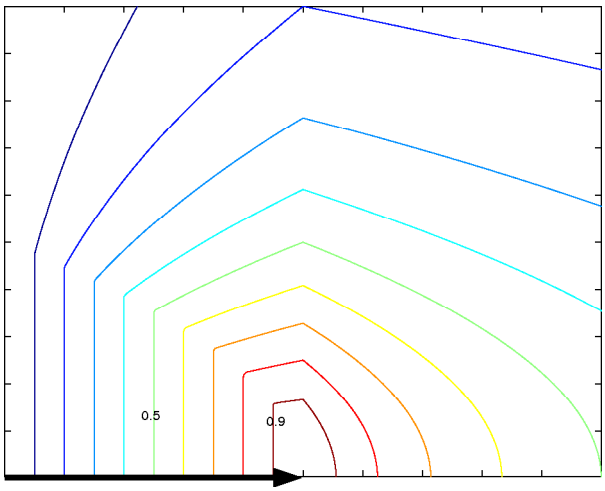


Figure 2.6: *Contour plot of the max measure. The maximum value is one, which is achieved if both vertices are identical. The main difference with the cosine measure is the exact way in which the possible external connections are measured.*

2.1.2 Row scaling

The previous measures only took possible external connections in account. But the exact number of external connections depends on the number of vertices in each net of the cluster (i.e., the number of non-zeros in the rows of the cluster). To scale with the exact number of external connections we apply row-wise scaling:

$$M(j,j') = \frac{1}{\omega(j,j')} \sum_{i=0}^{m-1} \frac{a_{ij}a_{ij'}}{f(\alpha_i)}. \tag{2.6}$$

If f is a monotonically increasing function, this puts extra weight on rows with only a few non-zeros. If there are for example only two non-zeros in one row, it is good to match these because then this row will not generate communication any more. We have investigated two different choices for f:

$$f_{\text{lin}}(\alpha_i) = \alpha_i, \tag{2.7}$$

$$f_{\text{exp}}(\alpha_i) = 2^{\alpha_i}. \tag{2.8}$$

Now remember that we can also see the matching as consisting of two (simultaneous) optimization problems; maximize the number of connections within a cluster, minimize the number of connections outside the cluster. The number of connections outside the cluster is simply the sum of the net degrees of the cluster. We achieve the simultaneous optimization by maximizing the number of internal connections divided by the number of external connections:

$$M(j,j') = \frac{1}{\omega(j,j')\Omega(j,j')} \sum_{i=0}^{m-1} a_{ij}a_{ij'}, \qquad \Omega(j,j') = \sum_{i:a_{ij}\neq 0 \vee a_{ij'}\neq 0} \alpha_i. \tag{2.9}$$

2.2 Some theoretical considerations

We are looking for a coarsening algorithm that will merge vertices that 'belong' in the same partition. In the previous sections we have proposed a method that uses an inner product measure to decide whether two vertices belong in the same partition. But can we also find a solid theoretical justification for this? The answer is yes.

We saw in section 1.2.2 that we can use the Laplacian of the hypergraph to partition the graph. We have to find a $\tilde{\mathbf{w}}$, $\tilde{w}_j \in \{-1,-1\}$, $\sum w_j = 0$ that minimizes $\tilde{\mathbf{w}}^T \mathcal{L} \tilde{\mathbf{w}}$. Vertices j, j' for which $\tilde{w}_j \tilde{w}_{j'} > 0$ belong to the same partition. Of course, we could explicitly calculate $\tilde{\mathbf{w}}$ and merge vertices for which $\tilde{w}_j \tilde{w}_{j'} > 0$, but this would be too expensive and would make the coarsening unnecessary since we already know the optimal partitioning. Still, we can use the Laplacian to find a theoretical justification.

We can explicitly write (1.15) as:

$$\sum_{i,j} \mathcal{L}_{ij}\tilde{w}_i\tilde{w}_j = \sum_i \nu_i - \sum_{i,j\in\mathcal{V}_-}\sum_l \frac{a_{li}a_{lj}}{\alpha_l} - \sum_{i,j\in\mathcal{V}_+}\sum_l \frac{a_{li}a_{lj}}{\alpha_l} + \sum_{i\in\mathcal{V}_-,j\in\mathcal{V}_+}\sum_l \frac{a_{li}a_{lj}}{\alpha_l} + \sum_{i\in\mathcal{V}_+,j\in\mathcal{V}_-}\sum_l \frac{a_{li}a_{lj}}{\alpha_l} \tag{2.10}$$

It is clear that to minimize the sum, the partial sums over $i,j \in \mathcal{V}_-$ and $i,j \in \mathcal{V}_+$ will have to be maximized. To achieve this we simply make sure that vertices i,j with the highest $\sum_l \frac{a_{li}a_{lj}}{\alpha_l}$ are merged. This sum is a row scaled inner product like the one proposed previously, with $f(\alpha) = \alpha$. Because we are using a greedy matching algorithm, we will have to find a matching order such that vertices with a potentially large term in the sum get to pick a match first. Vertices with large degrees and small net degrees would have to go first. We choose to sort the vertices only by their degrees, but it would also be possible to let vertices in small nets go first.

However, minimizing the Laplacian minimizes the wrong metric, namely the scaled communication volume. So, it remains to be seen if this will really give us better results.

2.3 Results

To compare the different similarity measures we have tested them on 18 matrices. Of course such a test set is quite arbritrary, but care has been taken to include matrices from different applications, like Markov chains, linear programming, finite differences, information retrieval, etc. In table 2.1 the matrices are listed. The test set contains the 15 matrices that are used in [33]. The parameters of the experiments are listed in table 2.2. We have compared the communication volume for partitioning into $P = 2, 4, 8, 16, 32, 64$ parts to the original Mondriaan results. All results are averages over 50 runs. Averages over all matrices for every P are presented, to see if certain methods work better for certain values of P. Averages over all matrices over all values of P are also presented. These are a bit crude, and a lot of information is lost in it. However, we are not tailoring Mondriaan to work well for a particular class of matrices, so a good method will have to perform well on average and not just for a few matrices. Please note that the averages are calculated over the scaled results.

#	**matrix**	m	n	nnz	**application**	**Struct. symm.**
1	`impcol_b`	59	59	317	Chemical engineering	n
2	`west0381`	381	381	2157	Chemical engineering	n
3	`well1850`	1850	712	8758	Geophysical surveying	n
4	`dfl001`	6071	12230	35632	Linear programming	n
5	`gemat11`	4929	4929	33185	Power flow optimization	n
6	`gemat1`	4929	10595	47369	Power flow optimization	n
7	`memplus`	17758	17758	99147	Circuit simulation	n
8	`cage10`	11397	11397	150645	DNA electrophoresis	y
9	`hyp_200_2_1`	40000	40000	200000	Laplacian operation	y
10	`onetone2`	36057	36057	227628	Circuit simulation	n
11	`cre_b`	9648	77137	260785	Linear programming	n
12	`tbdmatlab`	19859	5979	430171	Information retrieval	n
13	`finan512`	74752	74752	596992	Portfolio optimization	y
14	`lhr34`	35152	35152	764014	Chemical engineering	n
15	`nug30`	52260	379350	1567800	Linear programming	n
16	`bcsstk32`	44609	44609	2014701	Structural engineering	y
17	`bcsstk30`	28924	28924	2043492	Structural engineering	y
18	`tbdlinux`	112757	20167	2157675	Information retrieval	n

Table 2.1: *Test matrices. All matrices except 1, 3 and 6 are taken from the test of [33]*

#	**Matching order**				**Row-scaling**			**Match. id. first**
	decr.	rnd.	incr.	nat.	lin.	exp.	sum	
1	•							
2		•						
3			•					
4				•				
5	•							•
6	•				•			
7	•					•		
8	•						•	
9		•			•			
10		•				•		
11		•					•	
12			•		•			
13			•			•		
14			•				•	
15				•	•			
16				•		•		
17				•			•	

Table 2.2: *Experiment settings. All experiments are done with the no, cosine, min and max column-scaling. The sum row-scaling is the scaling from equation (2.9)*

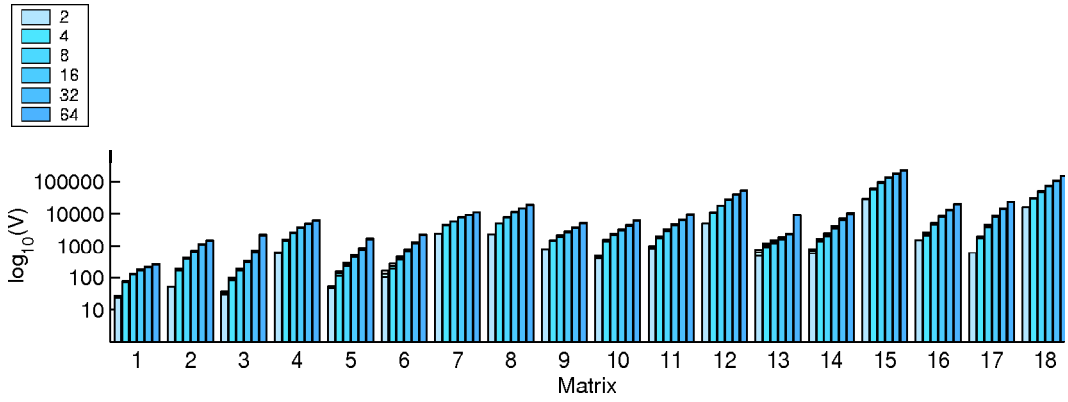


Figure 2.7: *Results of the original Mondriaan program for* $P = 2, 4, 8, 16, 32, 64$ *and* $\epsilon = 0.03$. *The default settings of Mondriaan v1.02 are used.*

matrix	**P**					
#	2	4	8	16	32	64
1	26	75	126	169	209	252
2	52	172	397	681	1113	1463
3	33	91	178	314	656	2158
4	601	1504	2582	3746	4954	6247
5	49	128	253	454	818	1653
6	139	241	386	737	1247	2232
7	2415	4562	5847	7704	9198	11145
8	2291	5022	7875	11194	14814	19162
9	800	1469	2018	2720	3691	5089
10	438	1503	2294	3148	4435	6343
11	907	1898	3066	4600	6565	9280
12	5004	10774	17764	28021	39525	52413
13	606	1042	1363	1735	2359	9309
14	695	1542	2222	3761	6668	10020
15	28590	58547	93929	130953	173984	221687
16	1528	2365	4928	8281	12946	19547
17	602	1836	4151	8372	14508	23339
18	15772	30411	49177	73140	106323	147114

Table 2.3: *Results of the original Mondriaan program. The true communication volume is listed.*

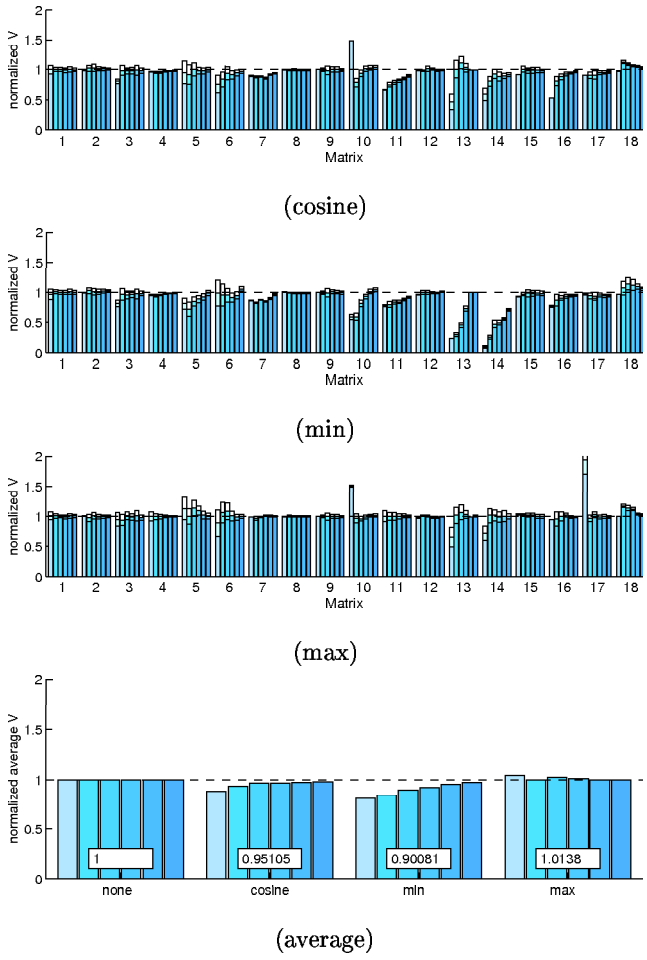


Figure 2.8: *Results of experiment 1. The matching order is decreasing weight. The cosine, min and max column-scalings are applied (no scaling is the default for Mondriaan v1.02). The results are normalized w.r.t the Mondriaan v1.02 results. The partitioning is done for* $P = 2, 4, 8, 16, 32, 64$ *and* $\epsilon = 0.03$. *The average over* 50 *runs is calculated over all matrices for every* P *separately and over all matrices, over all* P. *It is striking that the min measure works particularly well on matrices 13 and 14.*

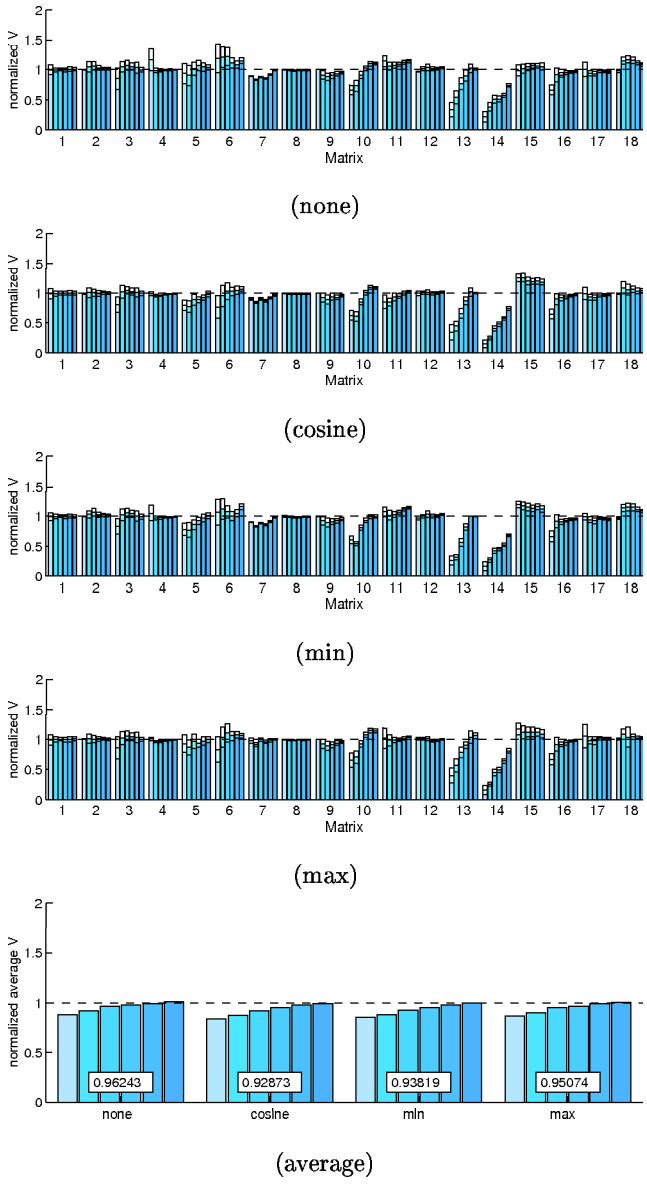


Figure 2.9: *Results of experiment 2. The matching order is random. It seems that random order works better than decreasing, on average. Again, matrices 13 and 14 do extremely well.*

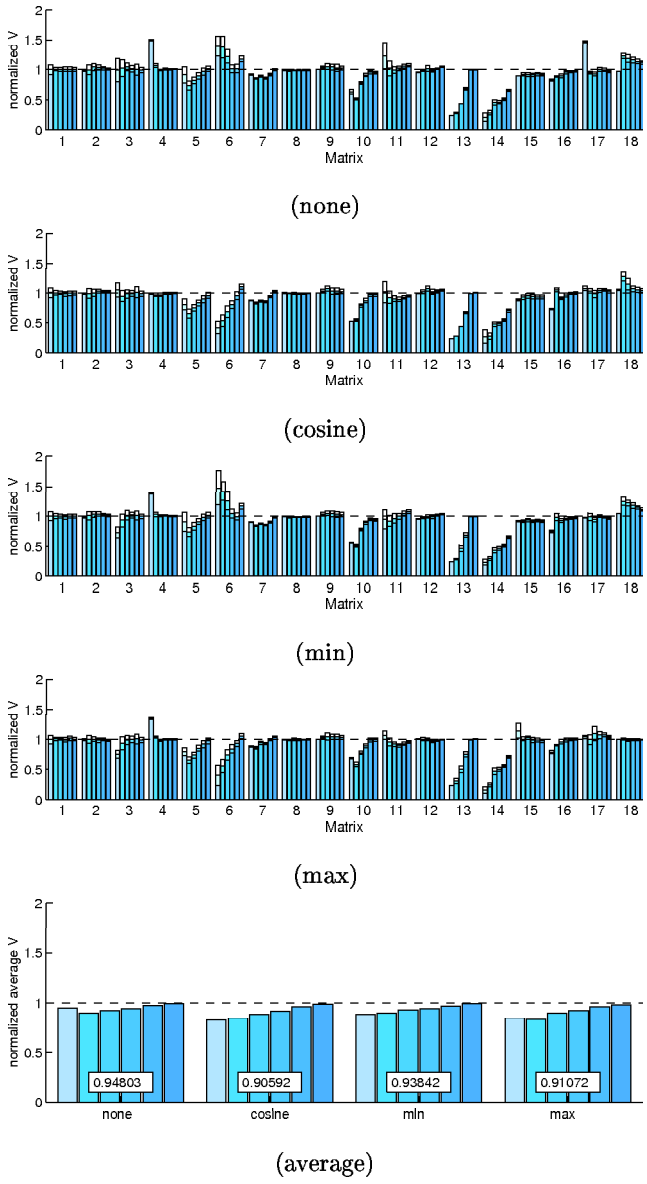


Figure 2.10: *Results of experiment 3. The matching order is increasing. This order does even better than random order on average. The cosine measure does best with this order. Matrices 13 and 14 stand out.*

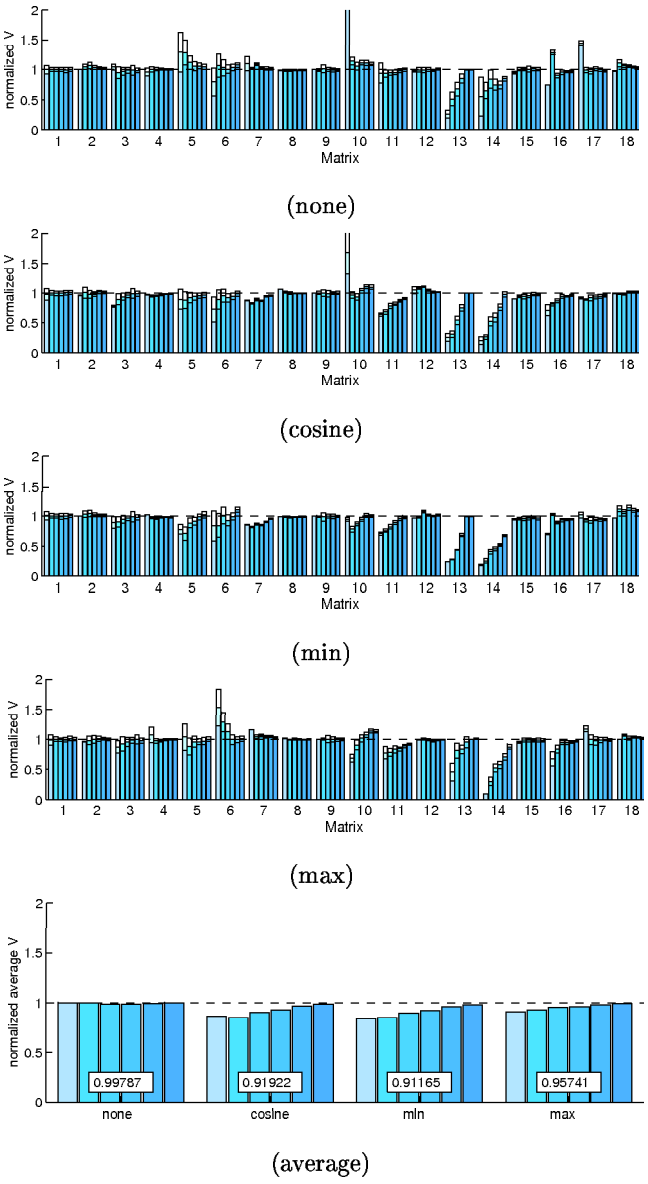


Figure 2.11: *Results of experiment 4. The matching order is natural. It seems that the natural order is not particularly beneficial on average.*

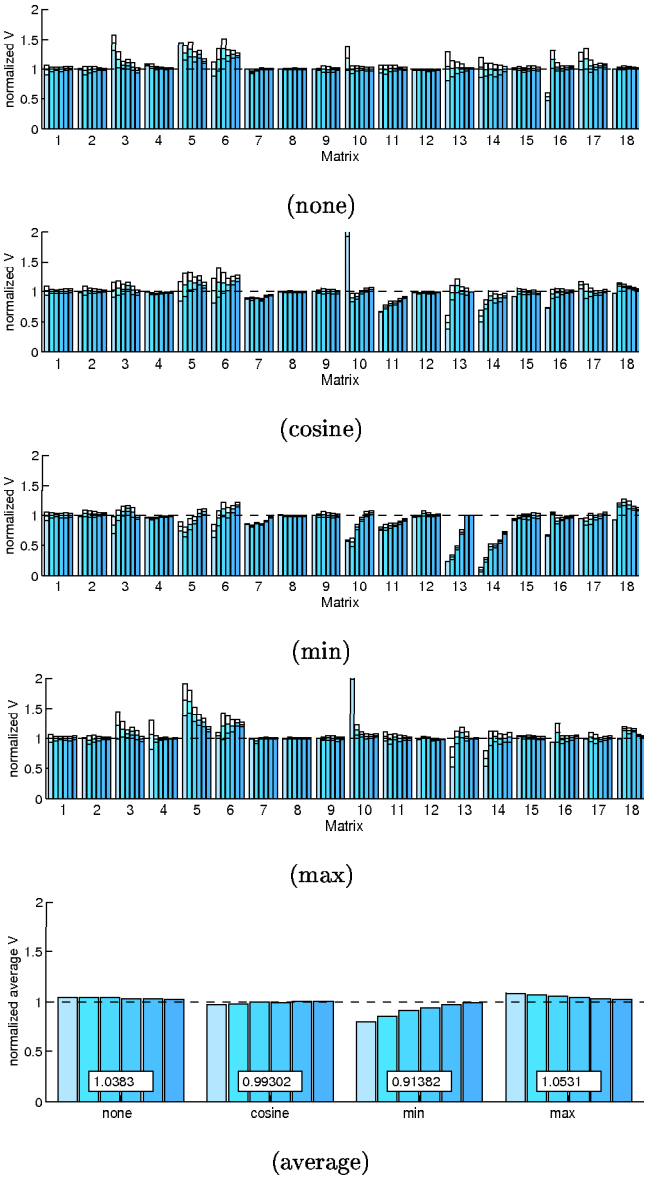


Figure 2.12: *The same settings as in experiment 1, but here the identical vertices are matched before any other matching is done. The results are a bit worse, and this is probably because there are less levels in the coarsening phase.*

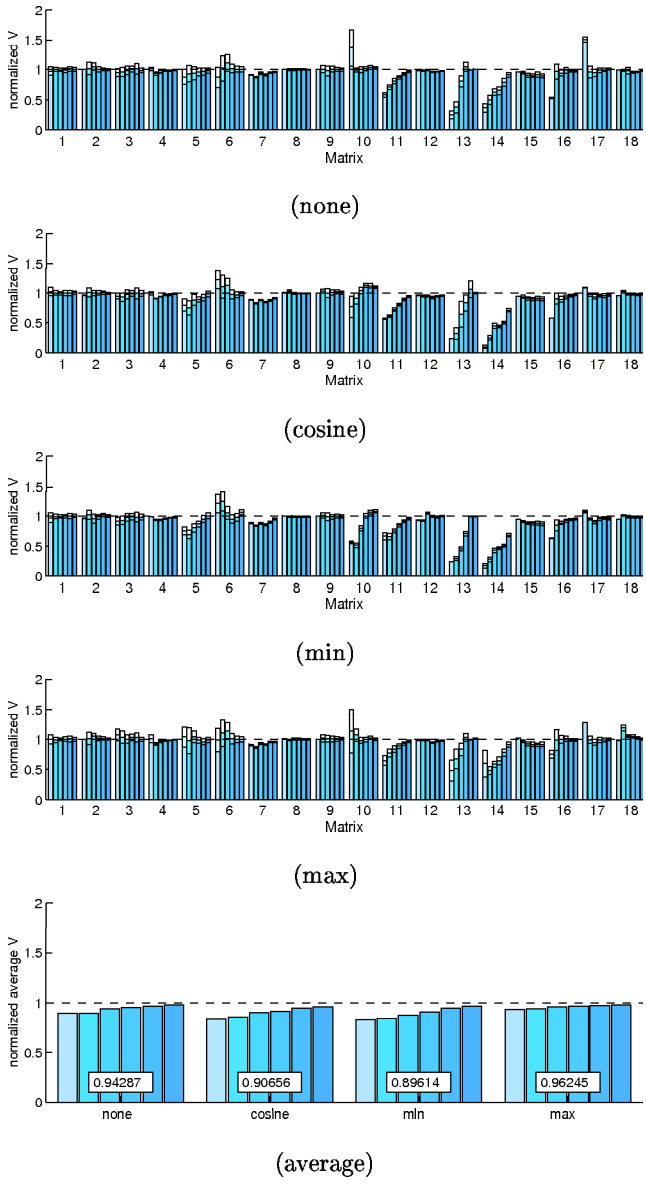


Figure 2.13: *Results of experiment 6. The matching order is decreasing, and linear row scaling is applied. This scaling works well, and combines well with column scaling.*

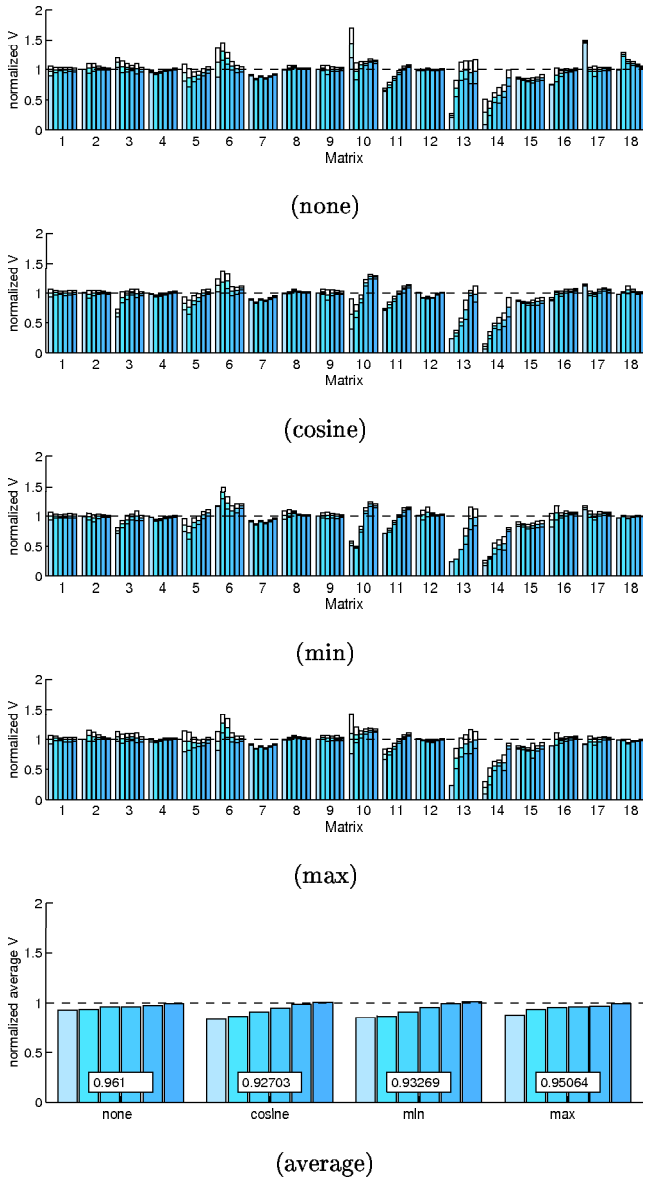


Figure 2.14: *Results of experiment 7. The matching order is decreasing, and exponential row scaling is applied. This row scaling does not work as well as the linear row scaling.*

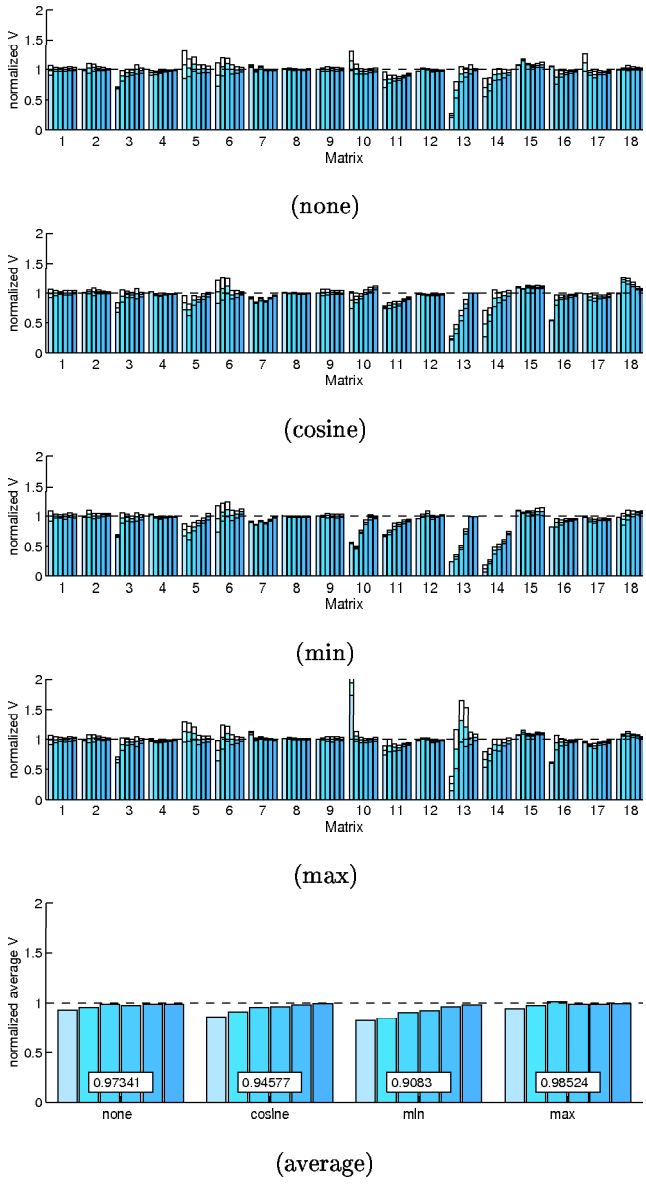


Figure 2.15: *Results of experiment 8. The matching order is decreasing, and sum row scaling is applied. This row scaling does not work as well as the linear row scaling.*

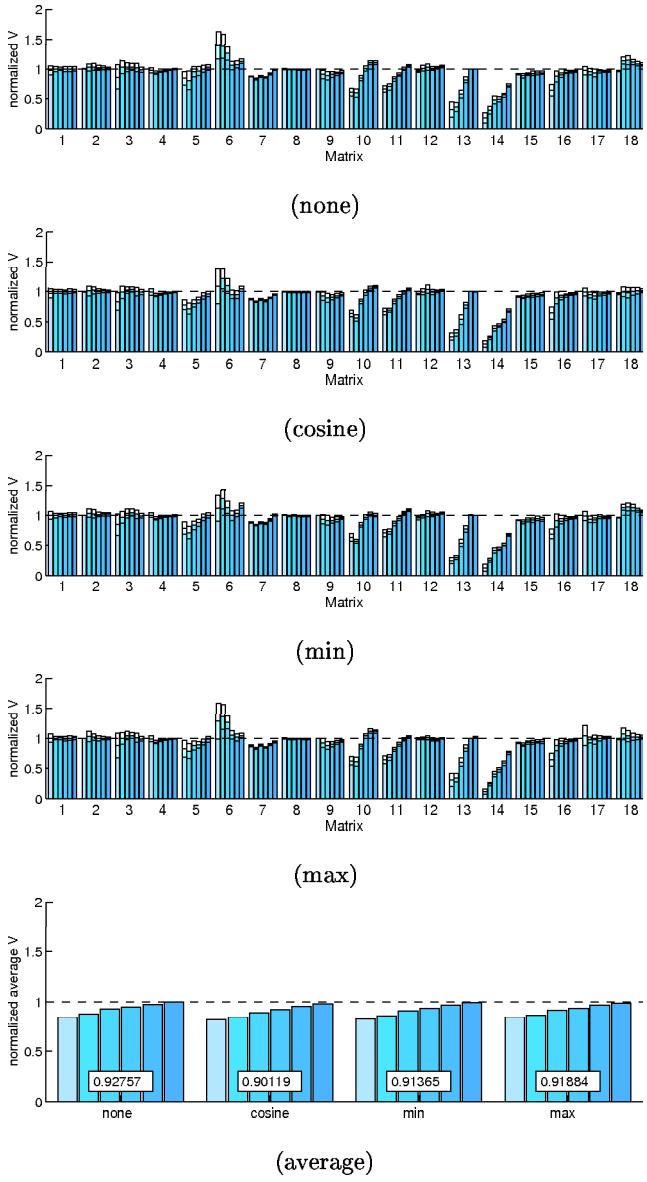


Figure 2.16: *Results of experiment 9. The matching order is random, and linear row scaling is applied. On average this works a little better than the linear row scaling with decreasing weight. This is not very surprising, since we already saw that random order was beneficial.*

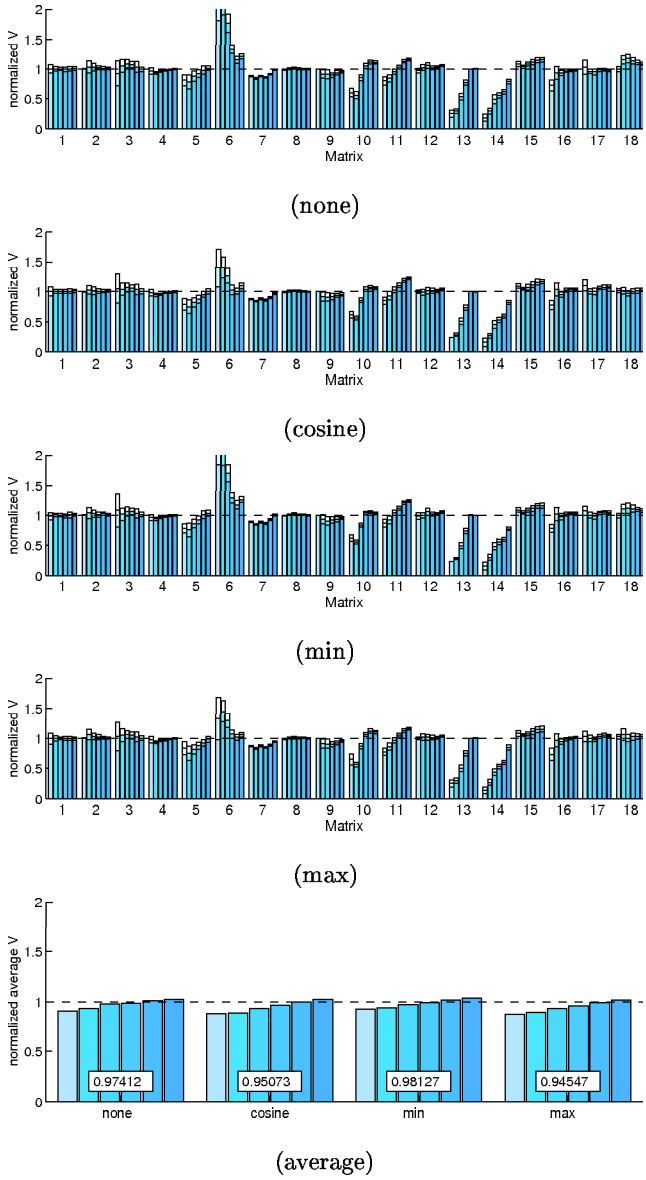


Figure 2.17: *Results of experiment 10. The matching order is random, and exponential row scaling is applied. The results are worse than those of the linear scaling. It seems that matching order also affects the row scaling.*

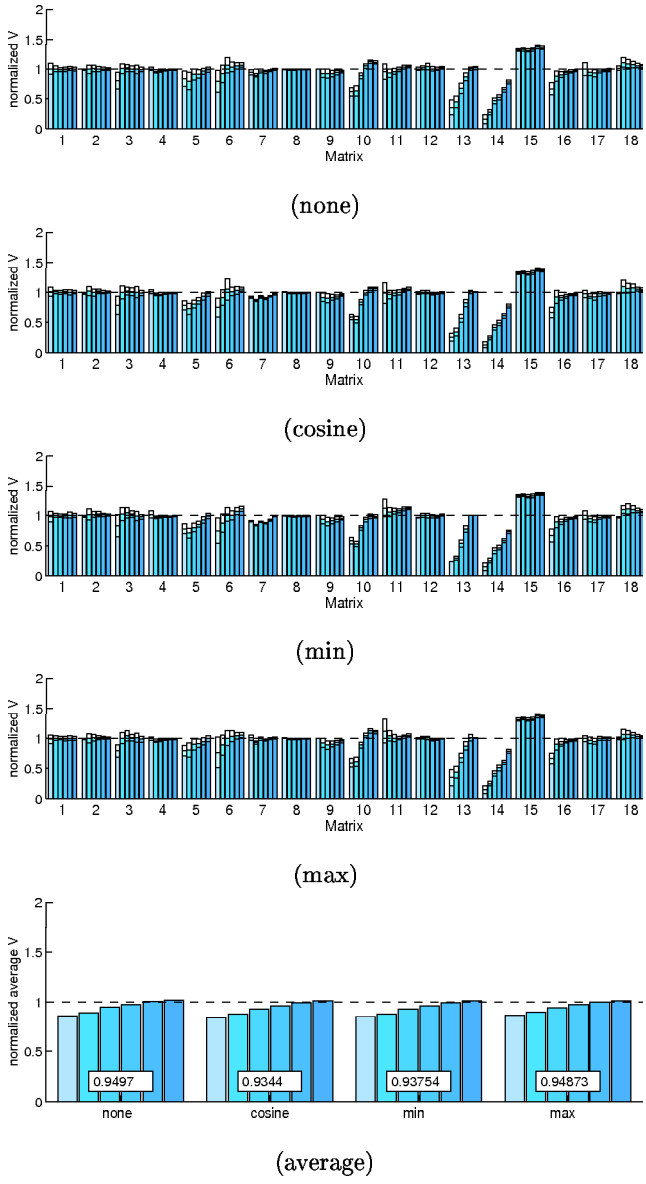


Figure 2.18: *Results of experiment 11. The matching order is random, and sum row scaling is applied. The results are a little better than the decreasing weight version (experiment 8).*

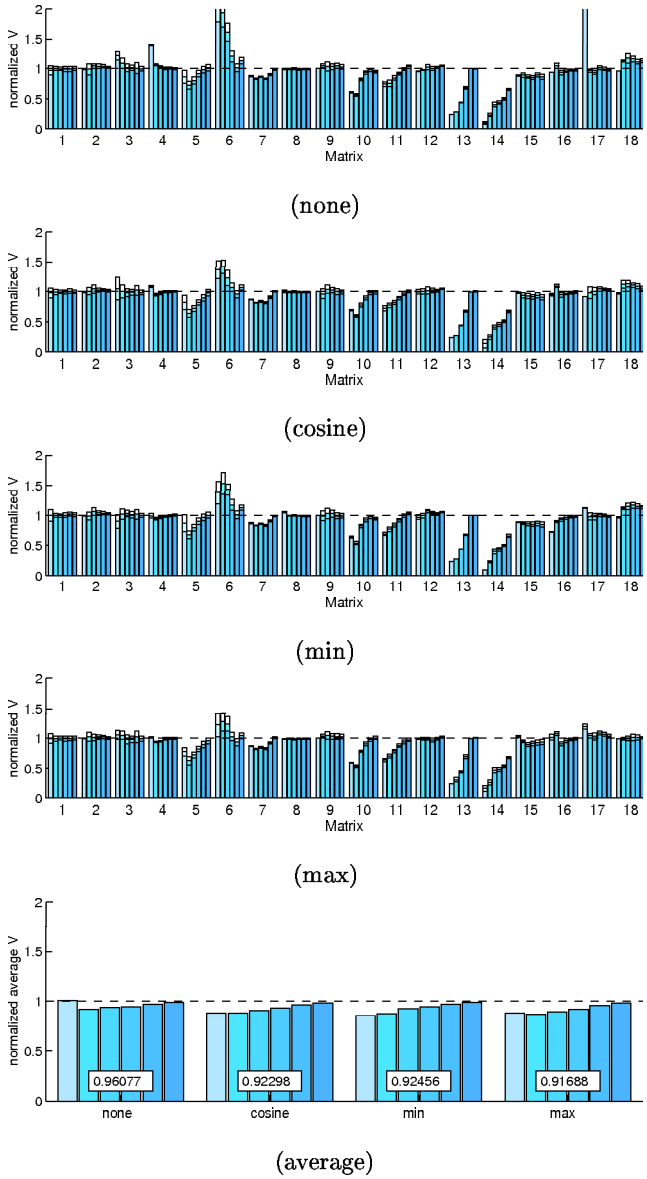


Figure 2.19: *Results of experiment 12. The matching order is increasing, and linear row scaling is applied.*

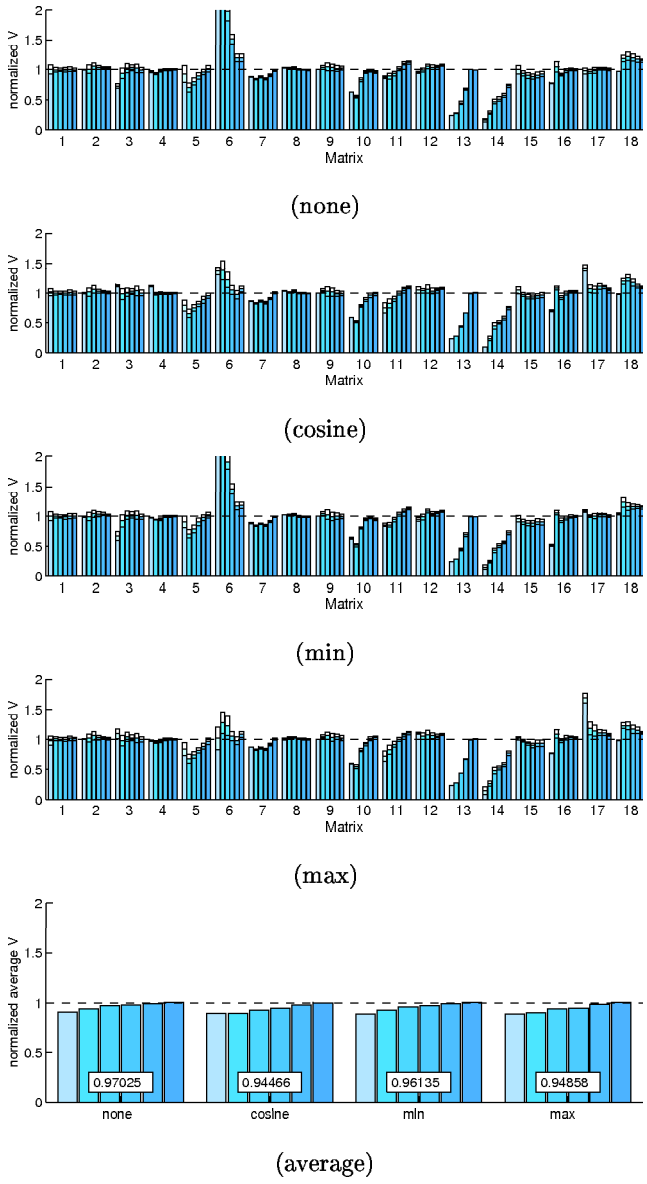


Figure 2.20: *Results of experiment 13. The matching order is increasing, and exponential row scaling is applied.*

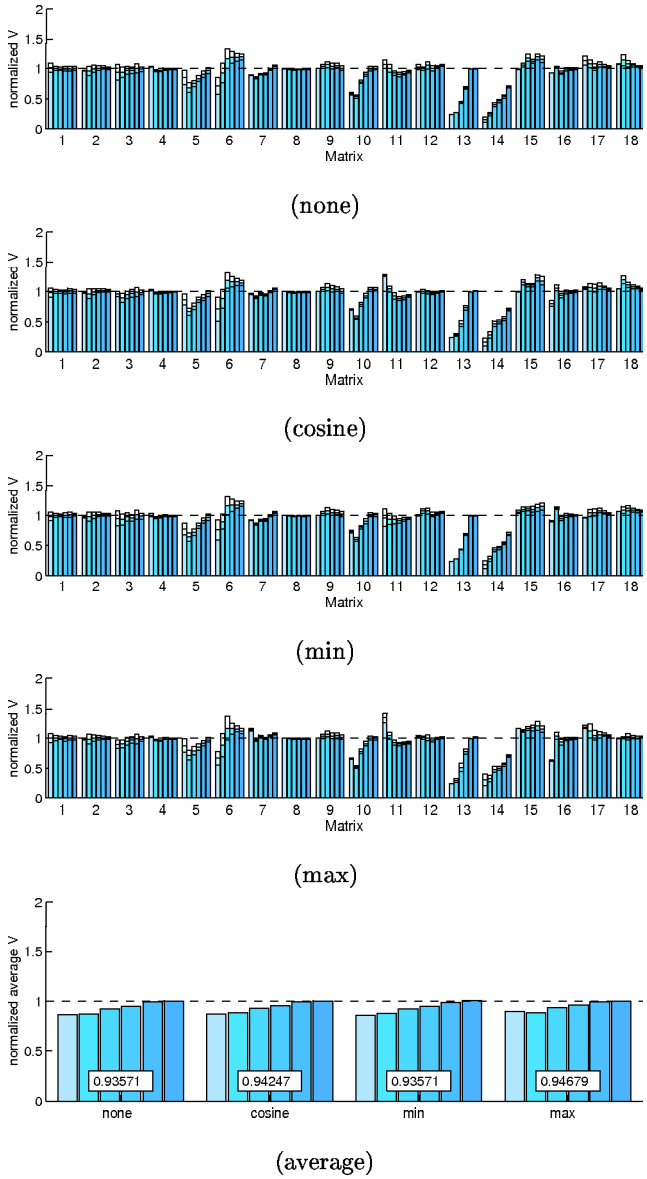


Figure 2.21: *Results of experiment 14. The matching order is increasing, and sum row scaling is applied.*

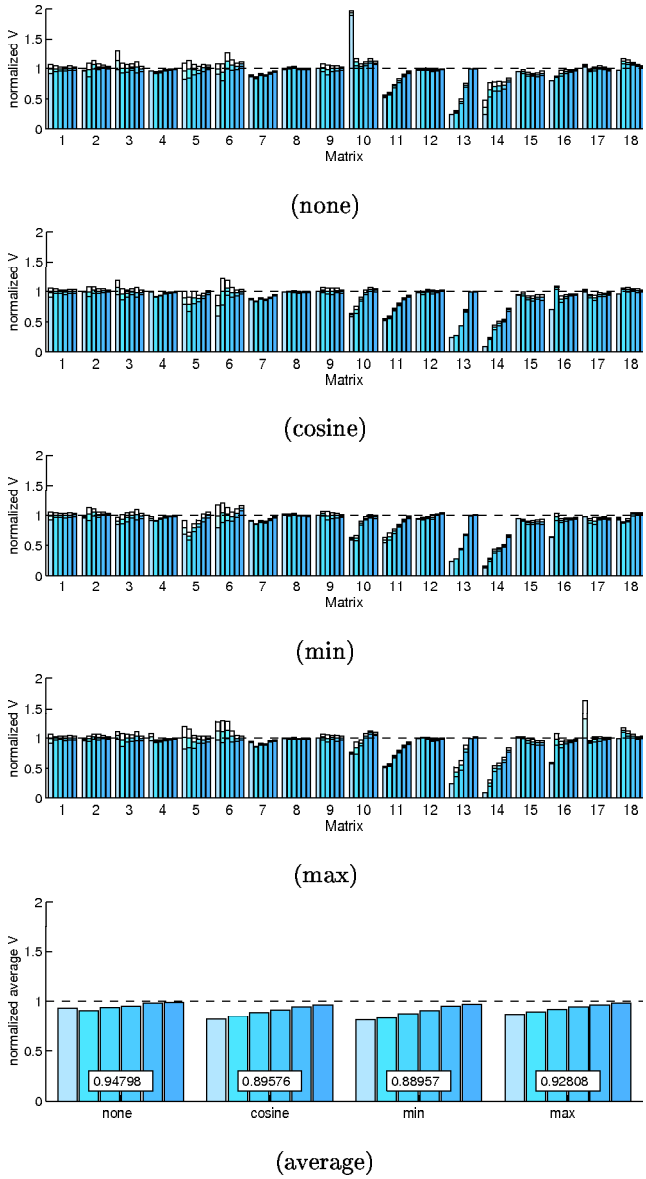


Figure 2.22: *Results of experiment 15. The matching order is natural, and linear row scaling is applied.*

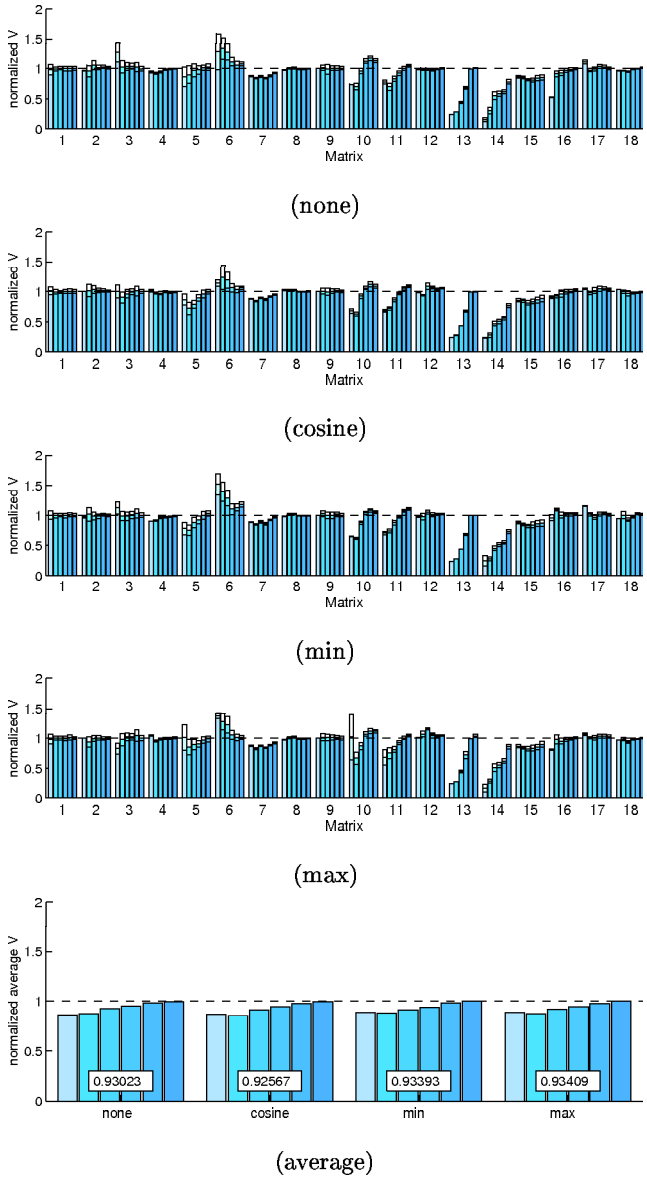


Figure 2.23: *Results of experiment 16. The matching order is natural, and exponential row scaling is applied.*

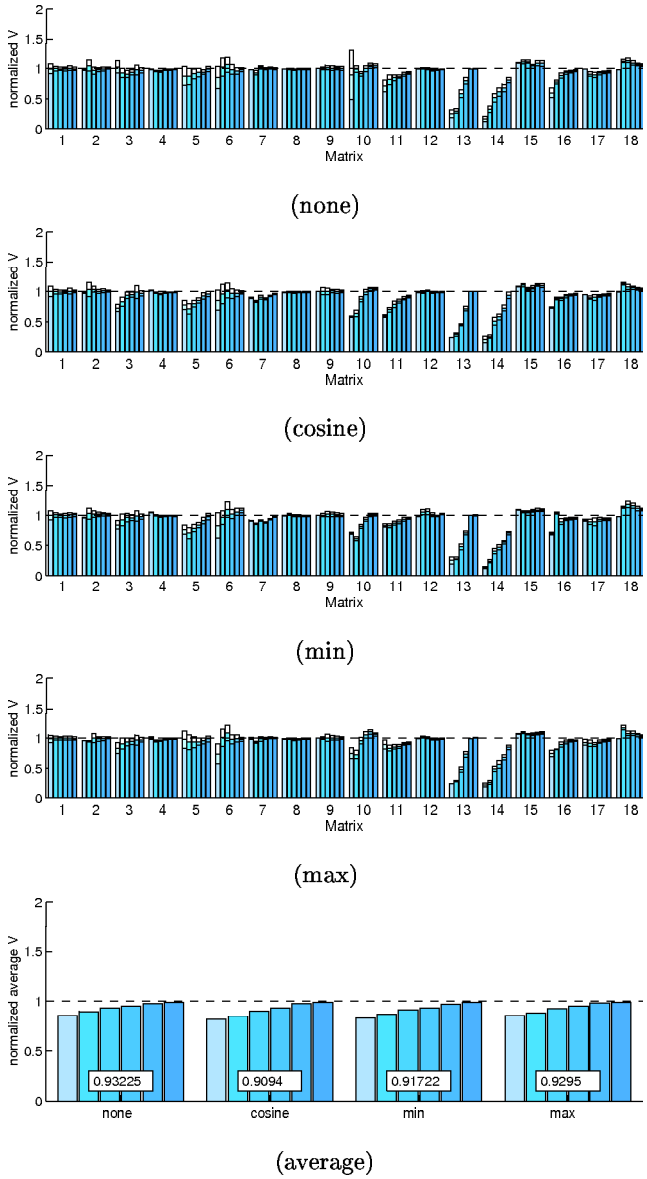


Figure 2.24: *Results of experiment 17. The matching order is natural, and sum row scaling is applied.*

2.4 Conclusion

We have seen that with column scaling and different matching orders an average reduction in communication volume of up to 11 % (up to 90 % for certain matrices) w.r.t to Mondriaan v1.02 is achieved. The row wise scaling also reduced the average communication volume is some cases, but the reduction is small compared to that of the column scaling. Certain matrices in our test set gave relatively much better results than others. We saw a reduction in communication volume for matrices 13 and 14 with most experiments, while matrix 6 often gave worse results. We did not find a scaling that gave better results for all matrices, although some came very close. In figure 2.25 all the averages are depicted. The best average results where achieved with: decreasing weight order, min column scaling, linear row scaling (experiment 6) and natural weight order, min/cosine column scaling, linear row scaling (experiment 15). All three gave an 11 % reduction on average. From these experiments we would advise to use one of these as new default values.

It is difficult to conclude from these experiments whether these scalings and matching orders are good for general coarsening algorithms. Here, we have not distilled the effect of just the coarsening on the overall quality of the partition. The refinement process is also of considerable influence, and we can we can never be entirely sure that the refinement did a good job thanks to, or despite of the coarsening. More elaborate experiments would be needed, measuring the quality of the coarsening. This could be done for example by seeing the coarsened graph as a partitioning of the original graph. The number of partitions would then be the number of vertices in the coarsened graph. The communication volume of this partitioning would be a good measure for the quality of the coarsening. To investigate the effect of the matching order, one could compare the results of the greedy algorithm to those where the full A^TA is used for the coarsening.

Another interesting experiment would be to continue with coarsening until the hypergraph is small enough to solve the optimal partitioning exactly. This would be expected to be better because of the increased number of levels, but also because there is no danger of getting stuck in a local minimum with KL. We have the optimal solution in the coarsest level, and we need only to maintain this during the refinement.

But, since the goal was to improve Mondriaan, and not to build a new partitioner from scratch, all these experiments are outside the scope of this thesis.

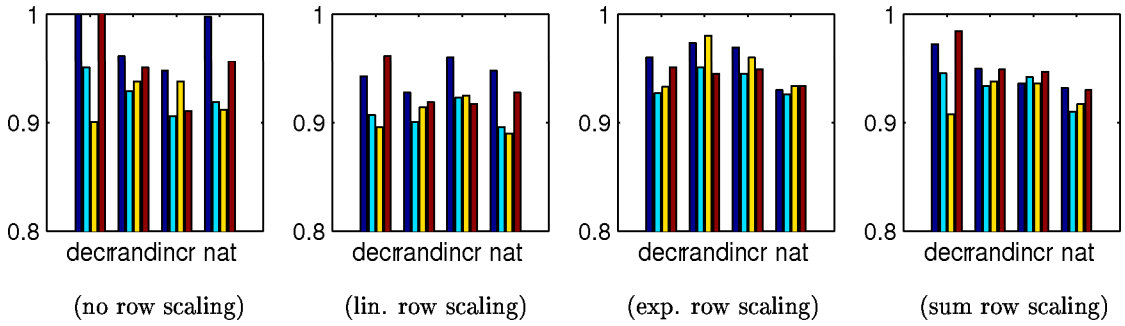


Figure 2.25: *Averages of all experiments. Decreasing weight combines well with the min scaling. The cosine scaling does best with increasing weight. Row scaling combines well with the column scaling. The linear row scaling seems to be a litter better than the other row scalings. The best average results where achieved with decreasing weight order, min column scaling, linear row scaling (experiment 6) and natural weight order, min/cosine column scaling, linear row scaling (experiment 15).*

Chapter 3

A fine-grained approach to partitioning

The Mondriaan package uses row- and column-net hypergraphs proposed in [9] to partition the matrix. Another hypergraph model was proposed in [11]. This is the so-called fine-grained hypergraph, where each non-zero is interpreted as a vertex, and the nets are defined by the rows and columns of the matrix. In experiments it has been shown that this approach could reduce the communication volume up to 50% compared to 1D partitioning. However, the partitioning was also considerably slower (up to 7 times as slow). Our goal here is twofold. First we would like to incorporate the fine-grained approach into the recursive bisection that is used in the Mondriaan package. Second, we would like to speed up the fine-grained partitioning. We will begin with pin-pointing the bottleneck in the fine-grained approach, and propose a solution to the problem. We will also discuss various ways to incorporate the fine-grained approach into Mondriaan.

3.1 The fine-grained approach

Given an $m \times n$ matrix A, having $\text{nnz}(A)$ non-zeros, a fine-grained hypergraph is defined as follows: Each non-zero a_{ij} represents a vertex, which is connected to all other vertices in row i by one net, and to all other vertices in column j by another net. This hypergraph has $\text{nz}(A)$ vertices and $m + n$ nets. Note that this hypergraph is considerably larger than the row-net or column-net hypergraphs. However, it has some nice structural properties. To visualize these we will introduce a matrix $\mathcal{F}_A$ associated with A. We drop the subscript from $\mathcal{F}_A$ if no confusion is possible. This matrix $\mathcal{F}$ is a sparse $(m + n) \times \text{nnz}(A)$ matrix whose row-net hypergraph is the fine-grained hypergraph of A. More specifically: Each column of $\mathcal{F}$ represents a non-zero a_{ij} of A. In this column of $\mathcal{F}$ there are 2 non-zeros: one at row i and one at row $m+j$. Thus, they

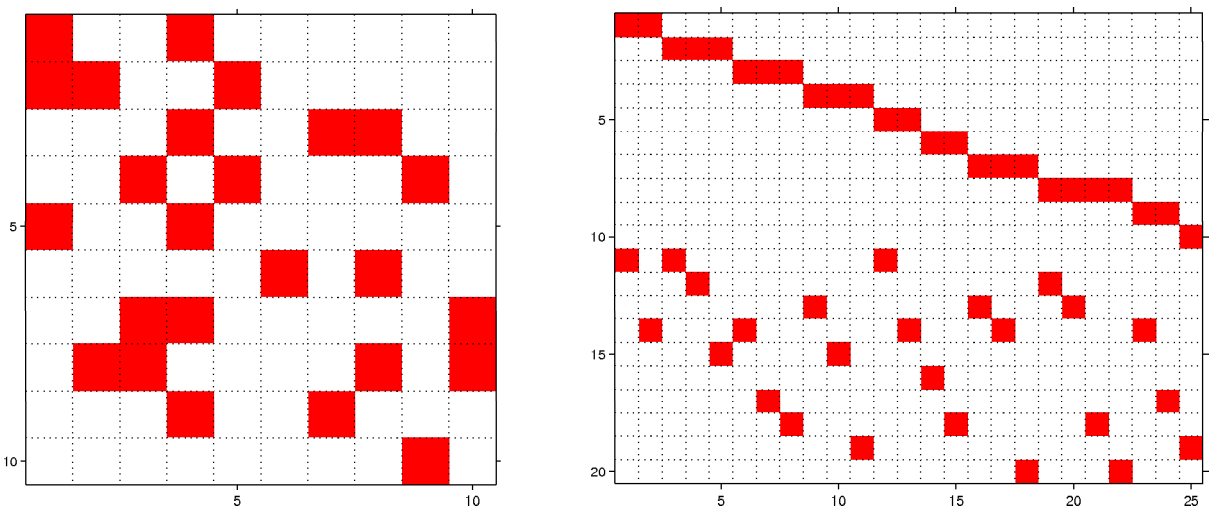


Figure 3.1: *Matrix and its $\mathcal{F}$ matrix. The row-net hypergraph of the $\mathcal{F}$ matrix is the fine-grained hypergraph of the original matrix. The row-nets of the fine-grained hypergraph are the top 10 rows of the $\mathcal{F}$ matrix and the column-nets are the lower 10 rows. This ordering is of course arbitrary.*

indicate the row and column of a_{ij}. There can be no identical columns in $\mathcal{F}$, since this would mean that the corresponding elements of A would be the same. This means that the inner product between two columns is either one or zero. Because of these special properties, the inner product matching is useless in the first stage. It just takes up a lot of time calculating ones. We will see if we can also find bounds for the inner product later stages.

3.1.1 Speeding up the matching stage

A straight forward way to cut the computation costs is to use random matching, where we match each vertex with a randomly picked adjacent vertex. In the first matching stage the inner products are either one or zero, so random matching is just as good as inner product matching. In later stages this random matching may not be so good, but because the vertex connections are rather sparse we may just get away with it.

When calculating the inner products, it is likely that there are several vertices that give the highest inner-product with the candidate. Since we assume that each of these vertices is as good as the other, we consider stopping the calculation as soon as such a vertex is found. All we have to do is find an upper bound on the inner product during the coarsening. As mentioned before, the maximal inner product is one in the first level. Now, each vertex has either two row nets and one column net or two column nets and one row net.

Thus, the maximal inner product in the second level is two. After a few stages, the maximum inner product is the degree of the vertex. to determine if the maximum inner product of a given vertex is its degree, we count the number of basis vertices (i.e., a vertex with one row net and one column net) needed to create the given vertex. If this is at least twice the number of basis vertices in the given vertex, than it is possible that an identical vertex has been made. In counting, we assume that each basis vertex only adds one extra row or column net to the given vertex. This way the number of basis vertices needed to create a vertex with n_r row nets and n_c column nets is $n_r \times n_c$. If the number of basis vertices in a given vertex is less than twice this number than we assume the maximum inner product is $\max\{n_r, n_c\}$. We will refer to this method as the 'ip1' method.

We also consider an alternating direction matching scheme. In the first stage we match each vertex with a vertex in the same row-net. Each vertex now has one row-net and two column-nets. The next stage we match vertices that have the same two columns nets. Each vertex now has two row- and two column-nets. By alternating, we keep the maximum inner product small. Also, the clusters will be more 2D in nature. We will refer to this method as the 'ip2' method.

3.1.2 Results

To measure the speed-up, the CPU times are measured of the different methods. The averages are calculated over 5 runs, and scaled with the time for the ip method. A profile is also given to determine the fraction of time spent on coarsening, initial partitioning, refinement and the rest. These are depicted in figure 3.2. The ip1 method is almost always faster then the normal inner product method, especially for the larger matrices where the coarsening takes up most of the time. For the smaller matrices, the partitioning and refinement take up most of the time, so no speed-up is expected there. Random matching, on the other hand, is sometimes considerably slower. We observe that with random matching, much more time was spent on the refinement phase. This is probably due to the fact that the matching itself was poor. It could be that for larger matrices, random matching is faster, because the coarsening overhead for the ip1 method is still considerable.

In figure 3.3 the scaled communication volumes for the fine-grained approach is depicted. The results are averages over 20 runs, scaled with the original Mondriaan results. The fine-grained approach does not always give better results than the 2D Mondriaan approach. The ip and ip1 methods work best on average.

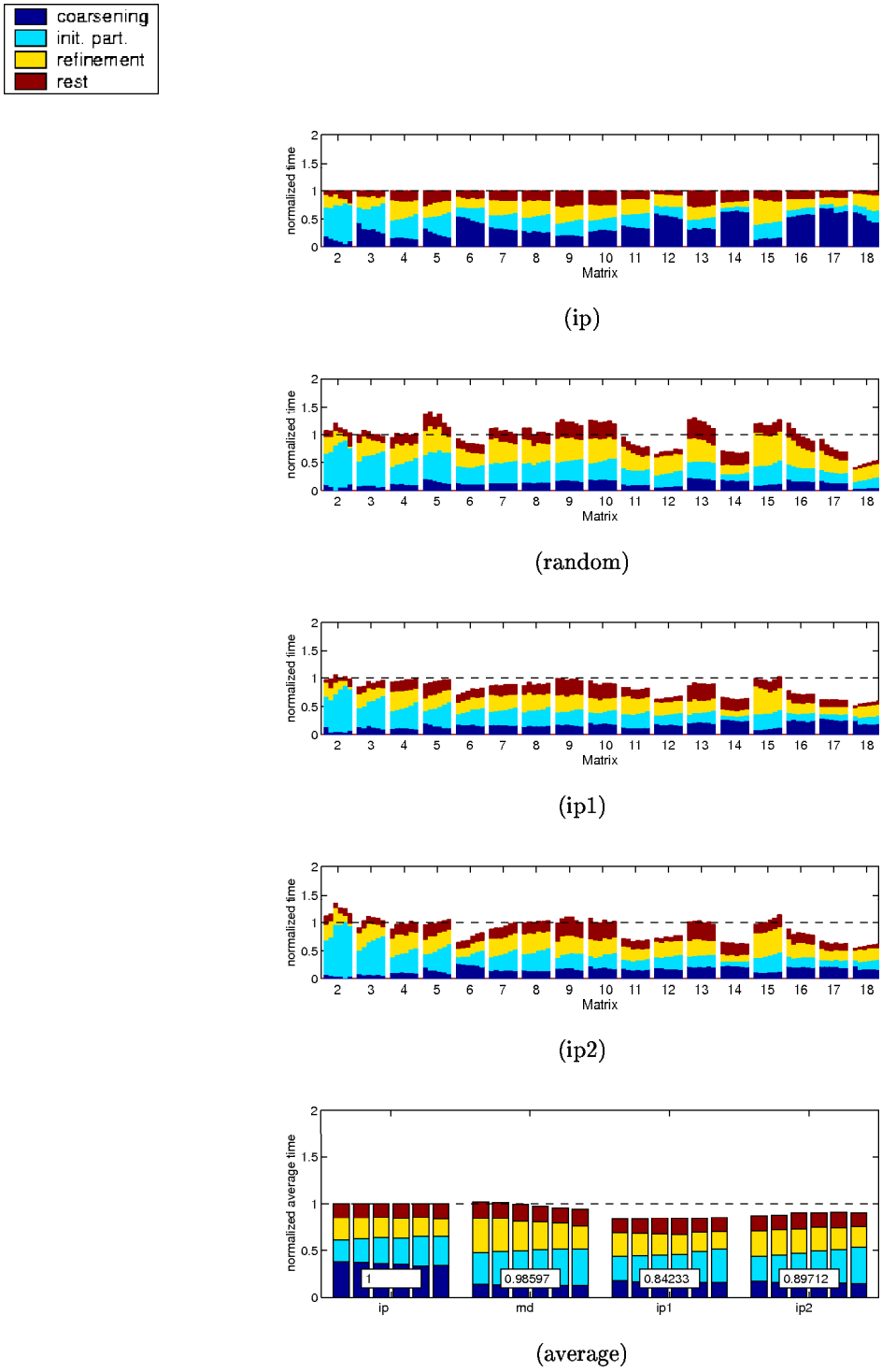


Figure 3.2: *Computation time of the different matching methods, scaled w.r.t inner product matching. The different colors indicate the different stages: coarsening, initial partitioning; refining and the rest.*

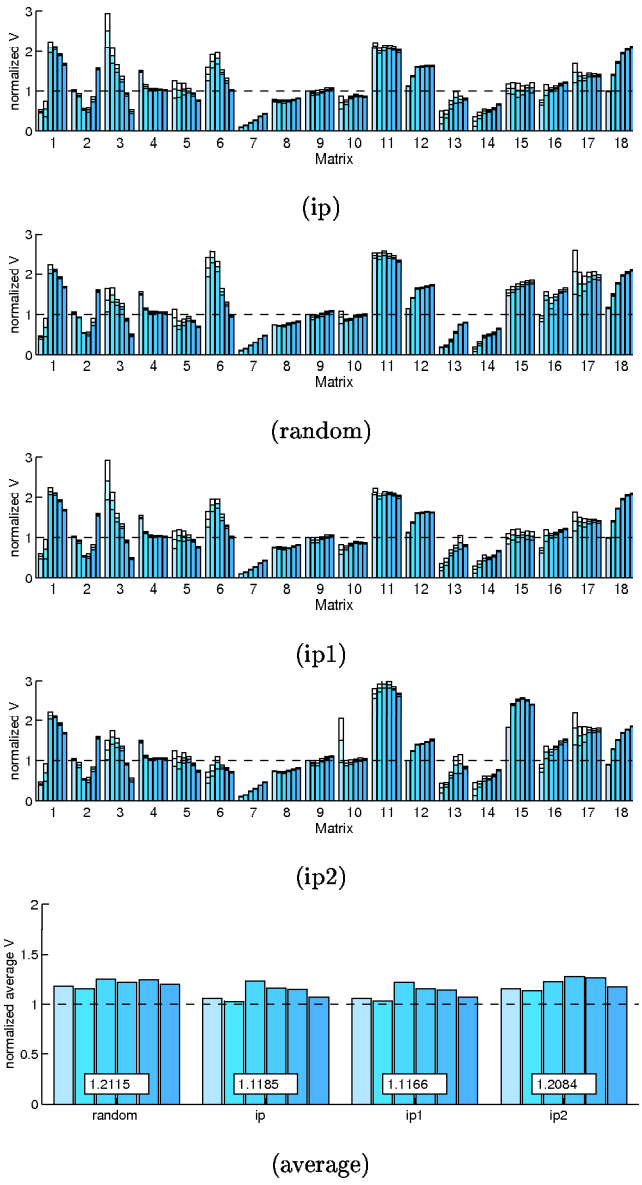


Figure 3.3: *Results with the fine-grained hypergraph method. Inner product matching works better than random matching, on average. The ip1 matching works just as good as inner product matching, while being faster. Alternating direction matching does not work so well on average, although it actually works better then ip for some matrices (matrices 3, 6).*

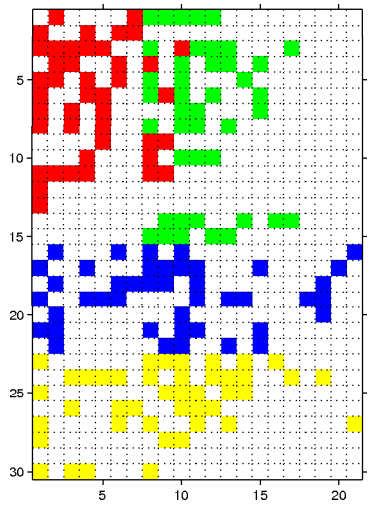


Figure 3.4: *Hybrid partitioning of a matrix into four parts,* $V = 41, \epsilon = 0.03$. *The first split is done row-wise, the second row-wise (bottom) and fine-grained (top).*

3.2 A hybrid method

Just as the original Mondriaan algorithm combined row-wise and column-wise partitioning, we would like to incorporate the fine-grained approach into the Mondriaan package, and combine the row-net, column-net and fine-grained hypergraph models. As we have seen, the fine-grained approach is not always an improvement. For successful incorporation we will need a metric to decide whether to use the fine-grained, the row- or column-net hypergraph model for a single bi-partitioning. See figure 3.4 for an example of such a distribution.

The original Mondriaan package has several ways of deciding between row- and column decomposition. The best, and most expensive, is to try both. Simply alternating the directions has also been tried, but the results where poor. A third option is to use the matrix dimensions as a criterion. Based on the communication volume for dense 1D bi-partitioning, we would choose row-wise partitioning if $m > n$ and column-wise partitioning otherwise. It seems that fine-grained partitioning works best for square matrices. Based on this observation, a simple criterion to choose between row, column, or fine-grained partitioning arises.

$$\begin{aligned} \text{if } m > \delta n &\rightarrow \text{row-wise partitioning} \\ \text{if } n > \delta m &\rightarrow \text{column-wise partitioning} \\ \text{else} &\rightarrow \text{fine-grained partitioning} \end{aligned}$$

We have tested taking the best of three partitionings (row, column and fine-grained), and the above criterion for various values of δ.

3.2.1 Results

In figure 3.5 the results of the hybrid methods are depicted. The 'best' hybrid method successfully combine the three hypergraph models.In most cases, the communication volumes are lower than or equal to those of the original Mondriaan algorithm. For some matrices the communication volumes are a bit higher. In these cases, the later bi-partitionings suffered from an earlier choice to do a fine-grained partitioning. Using the matrix dimension as a criterion to choose is not very beneficial on average.

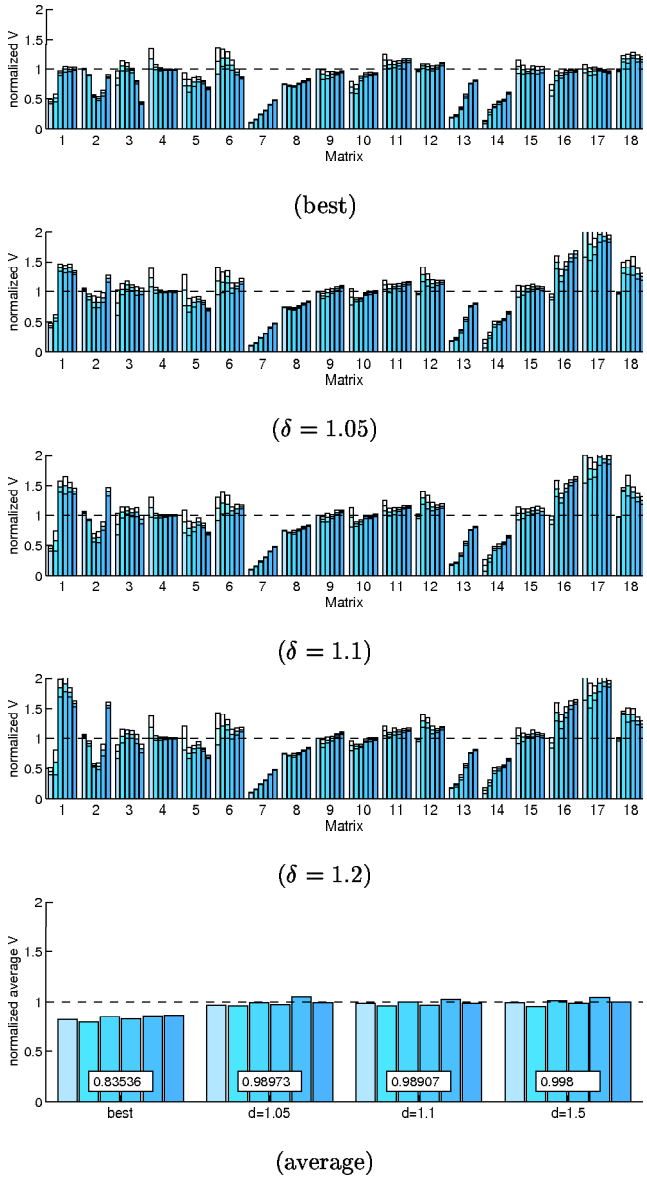
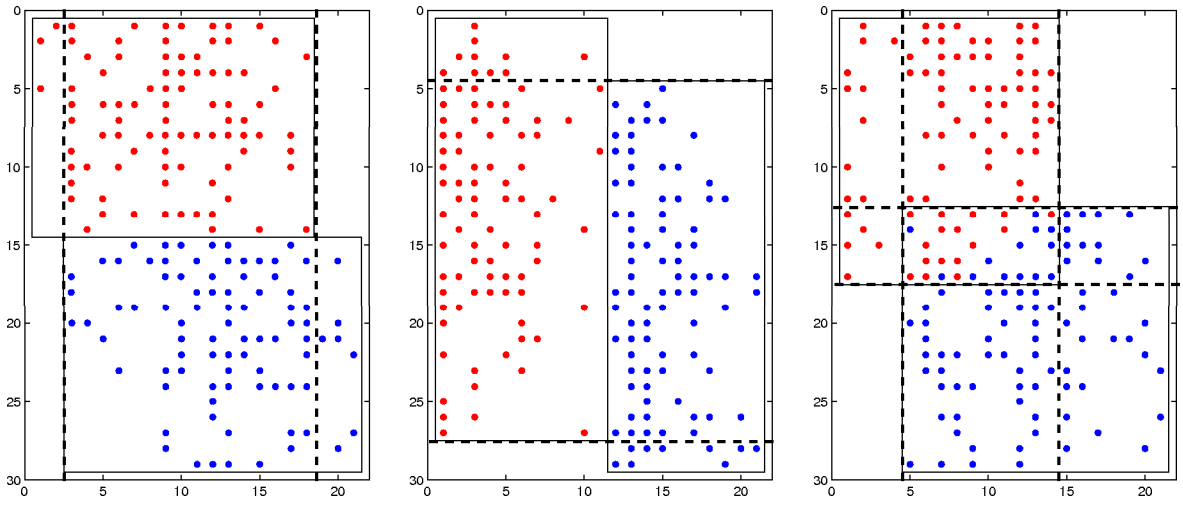


Figure 3.5: *Results with the hybrid method method. The results are averages over 20 runs, and scaled with the original Mondriaan results. For the row/column hypergraphs decr. order and HCM matching is used, for the fine-grained hypergraphs random matching is used.*



(a) (b) (c)

Figure 3.6: *Permutations of partitioned matrices with $P = 2$. The permuted matrix should be as close to block diagonal form as possible. The area between the dotted lines generate communication. (a) is a row partitioning, (b) is a column partitioning and (c) is a fine-grained partitioning. It is clear that there is much more freedom with the fine-grained approach to obtain a good partitioning.*

3.3 Conclusion

We have seen that the fine-grained approach yields better results for some matrices, and much worse for others. In theory, the fine-grained approach should always give better results, since the solution space is larger. In figure 3.6 we see an example of a fine-grained partitioning, and how the extra freedom can be interpreted. Also, the fine-grained hypergraphs are much larger, so there are more refinement levels. This may also explain why the results with the fine-grained approach are sometimes better. It may well be that a fine-grained partitioning is in fact a row or column wise partitioning. To investigate this, we look at matrices 1 (figure 3.7), 7 (figure 3.8), and 8 (figure 3.9). The fine-grained method gave much better results than the 2D Mondriaan approach with these.

We conclude that in some cases the partitioner was able to benefit from the extra freedom the fine-grained approach has. The gains where not purely due to the extra refinement levels. In some cases, however, the communication volumes where much higher with the fine-grained method.

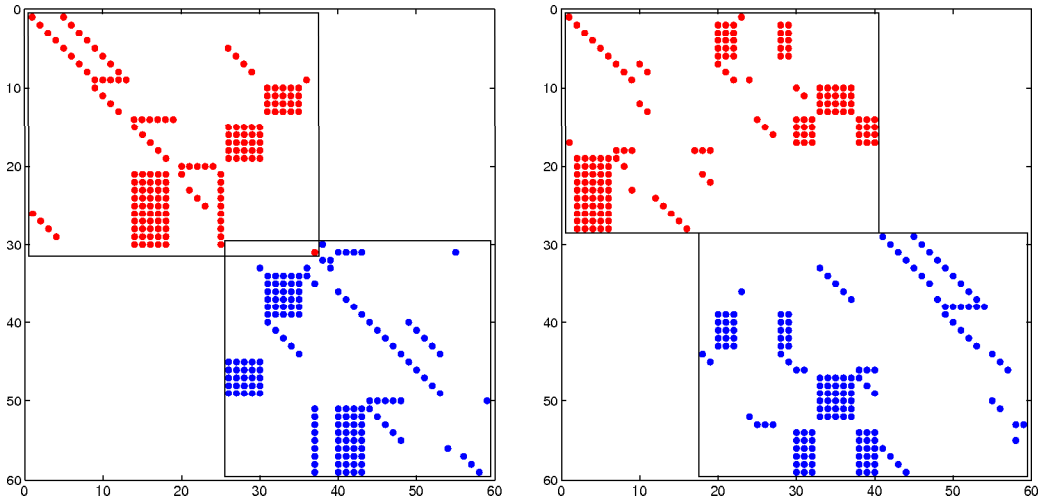


Figure 3.7: *With matrix 1 (*`impcolb`*), the fine-grained partitioning ($V = 14$) is almost a row partitioning. Please note that there was no refinement in the row partitioning ($V = 22$). The fine-grained hypergraph is larger, and did have several levels. This is probably the cause of the lower volume of the fine-grained partitioned matrix.*

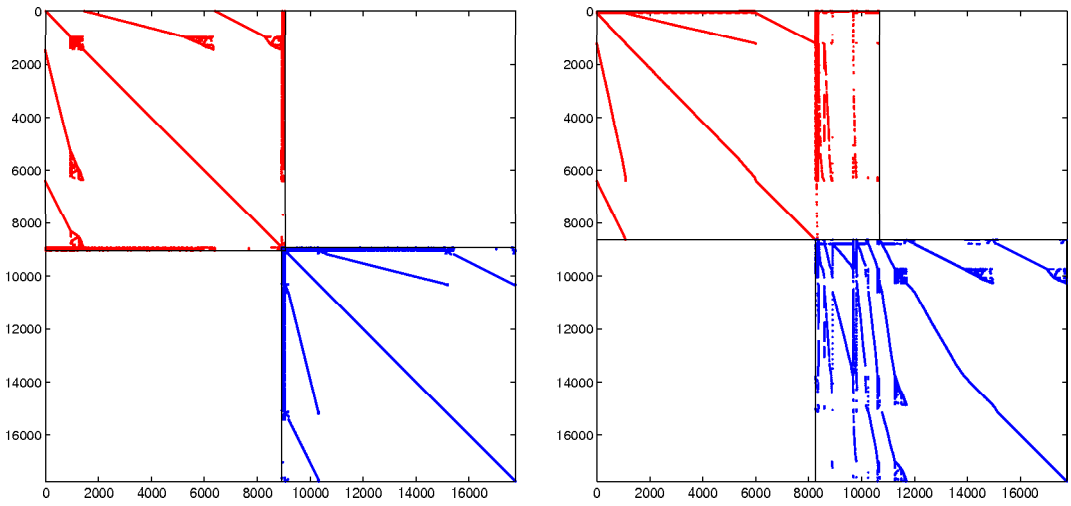


Figure 3.8: *For matrix 7 (*`memplus`*), the fine-grained partitioning (a), is really a 2D partitioning with $V = 255$. The row partitioning (b) has $V = 2410$.*

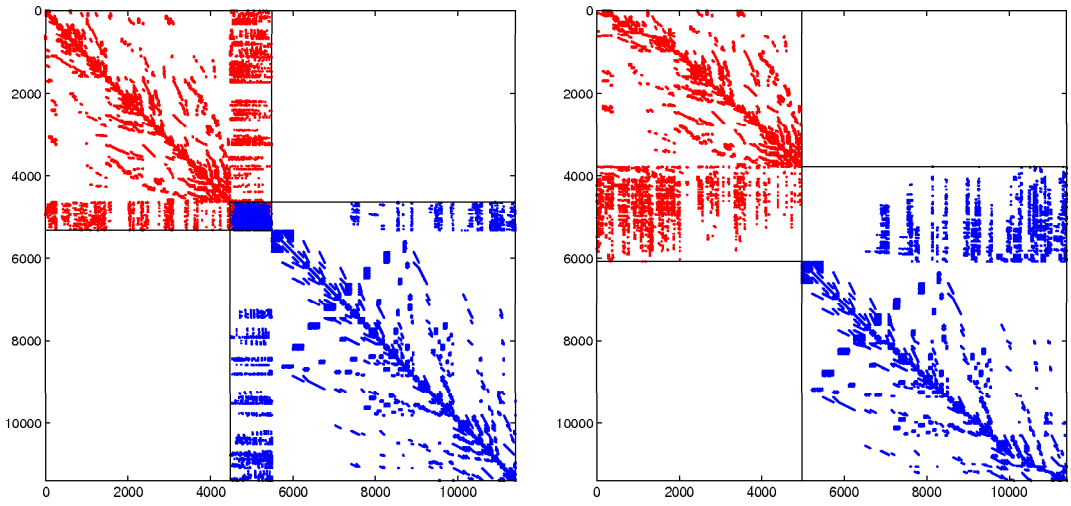


Figure 3.9: *For matrix 8 (*`cage10`*), the fine-grained partitioning (a), is really a 2D partitioning with* $V = 1710$. *The column partitioning (b) has* $V = 2280$.

Chapter 4

Comparing different 2D partitioning strategies

To partition a matrix, two different, hypergraph-based, 2D partitioning strategies have recently been proposed [33, 11]. The Mondriaan package uses the 1D hypergraph model, and chooses the best split direction in each bi-section. The fine-grained hypergraph model is a more general way to obtain a 2D partitioning. Both methods have been shown to work better than 1D hypergraph based partitioning. In Chapter 3, we saw that in some cases the 2D Mondriaan strategy gave better results than the fine-grained strategy and vice versa. In this chapter, we want to perform a thorough comparison of the two 2D partitioning methods and the hybrid method, proposed in Chapter 3. To do this we use the Mondriaan package, and a modified version of Mondriaan that uses PaToH [10] to do the actual hypergraph partitioning. We do this, because it may well be that the Mondriaan hypergraph partitioner favors row/columnnet hypergraphs. By using another hypergraph partitioner we hope to remove any possible bias, and be able to draw more general conclusions on the methods involved.

4.1 Results

In figures 4.1 and 4.2, the results of the experiments are presented. The results are scaled with the original Mondriaan results. We have already seen that the fine-grained method was not always an improvement. This is confirmed by the experiments using PaToH. The hybrid method works best with both partitioners, although the gain w.r.t the 2D Mondriaan approach is much smaller for PaToH.

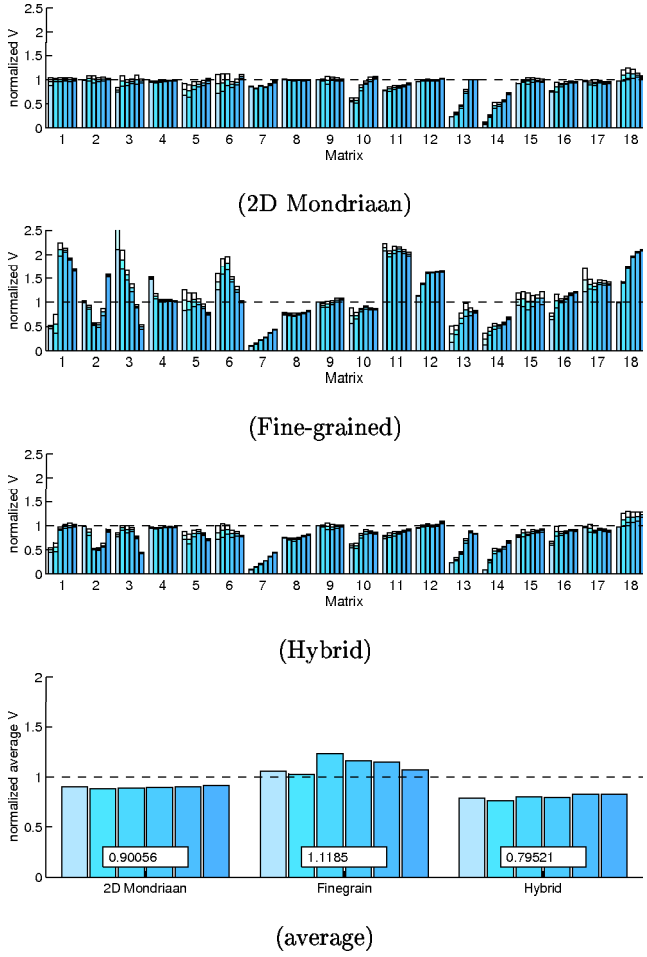


Figure 4.1: *Results of the three different 2D partitioning methods, using Mondriaan's own hypergraph partitioner, with decr. weight, min column scaling for row/column hypergraphs, and inner product matching for the fine-grained hypergraph. The results are scaled with the original Mondriaan.*

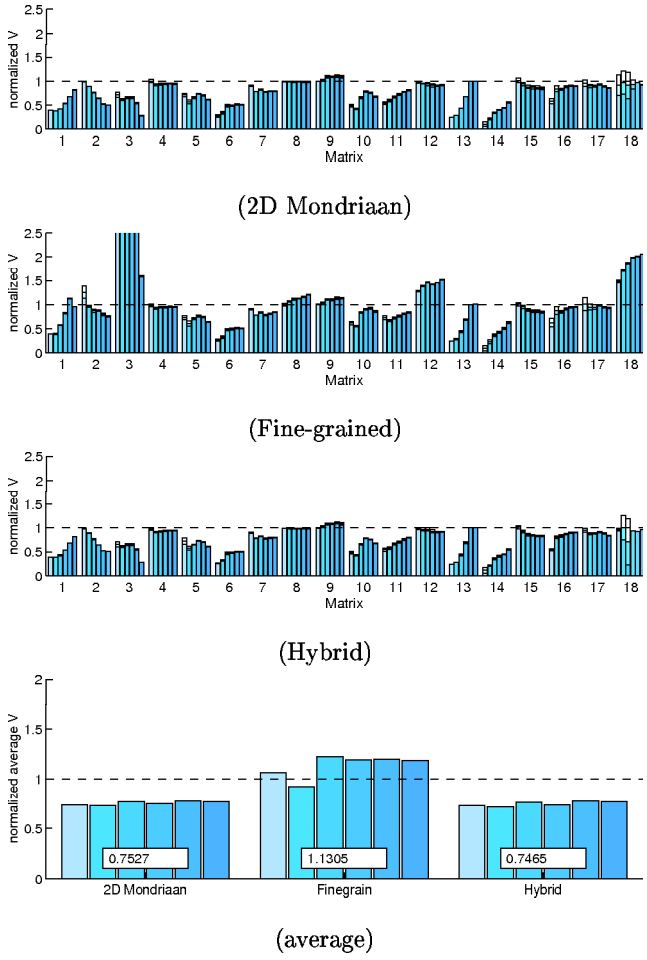


Figure 4.2: *Results of the three different 2D partitioning methods, using PaToH as hypergraph partitioner. The results are scaled with the original Mondriaan.*

4.2 Conclusions

We sought to compare two different 2D partitioning methods; 2D Mondriaan and fine-grained. Both methods have been shown to give much lower communication volumes than 1D partitioning [33, 9]. In our experiments, we see that fine-grained is not always better than 2D Mondriaan. When using PaToH as a hypergraph partitioner, the fine-grained method gave communication volumes equal to, or higher than the 2D Mondriaan method. With the hybrid method the fine-grained approach was rarely chosen. Using Mondriaan's own partitioner, the hybrid method did prove an improvement, but the average communication volume was still higher than that of PaToH. Apparently Mondriaan's hypergraph partitioner does, in some cases, benefit from the extra freedom that the fine-grained hypergraph gives. An important factor here is probably the higher number of levels in the coarsening phase.

We have shown here that it is possible to obtain communication volumes with the 2D Mondriaan method that are equal to, or lower than those of the fine-grained method. However, the results depend strongly on the hypergraph partitioner used. So, it is possible that when using a hypergraph partitioner tailored for fine-grained hypergraphs, the fine-grained method will give better results. Because the solution space is much larger with the fine-grained model, better results can be obtained in theory. But it seems that the hypergraph partitioners tested here cannot always handle this extra freedom well, and get stuck in a local minimum.

Chapter 5

Test case: partitioning PageRank matrices

In recent years a lot of effort has been made to develop better and faster search engines for the web. A very successful and widely used method to find relevant pages is the PageRank metric [29], which is at the heart of the Google search engine. Basically, a Markov chain is constructed from the webgraph and a user-centered model. The webgraph represents the link structure of the world wide web. The power method (see, for example [16]) is then used to get the largest eigenvector of this Markov matrix. Because these webgraphs can be enormous, a parallel PageRank algorithm is needed. Such a parallel algorithm is proposed in [7]. It is no surprise that this algorithm is dominated by parallel matrix-vector multiplications. So, partitioning these very large PageRank matrices is an important application for matrix partitioners like Mondriaan. In this chapter we would like to test the improvements made to Mondriaan in this thesis on two PageRank matrices, and compare the results with earlier results on these particular PageRank matrices from [7], where the authors use the parallel hypergraph partitioner Par*k*way [31] with the fine-grained hypergraph model. The characteristics of the PageRank matrices used are listed in table 5.1. The results are listed in table 5.2

Matrix	m	n	nz
`Stanford`	281903	281903	2594228
`Stanford_Berkeley`	683446	683446	8262087

Table 5.1: *Properties of the PageRank matrices.*

	Stanford					
P	2	4	8	16	32	64
Par*k*way 1D	n.a.	13849	34221	74137	n.a.	n.a.
Par*k*way 2D	n.a.	**1399**	**2285**	**4307**	n.a.	n.a.
Mon. none	1223(1032)	2770(2205)	4015(3309)	7171(5840)	10557(9630)	17171(16305)
Mon. min	**709**(628)	1941(1629)	3214(3024)	6928(6650)	10406(9365)	16676(16265)
Mon. cosine	731(475)	1752(1373)	3673(3030)	6029(5784)	**9651**(**9072**)	**16463**(**15670**)
Mon. min/lin	526(526)	1579(1470)	3753(3469)	6194(**5716**)	9905(9469)	16820(16389)
Mon. cosine/lin	726(**425**)	1528(**1201**)	3196(**2946**)	6658(6052)	10027(9079)	17011(16792)
Mon. none/lin	1306(1001)	1881(1475)	3540(2895)	7333(6704)	11030(10169)	17978(17625)
	Stanford_Berkeley					
P	2	4	8	16	32	64
Par*k*way 1D	n.a.	6648	45132	147590	n.a.	n.a.
Par*k*way 2D	n.a.	2081	**3479**	**7302**	n.a.	n.a.
Mon. none	769(642)	**1852**(1569)	5269(3483)	11292(10142)	19577(18132)	31873(28758)
Mon. min	1028(1027)	2539(2162)	4245(3753)	10797(9661)	17496(16502)	29215(27464)
Mon. cosine	746(590)	2143(1749)	4088(3519)	9604(9174)	16416(14866)	27891(26749)
Mon. none/lin	760(565)	2720(1942)	5096(3620)	10893(10350)	18300(18131)	29718(29003)
Mon. min/lin	**664**(649)	2722(2182)	5264(3557)	9424(**8146**)	**15239**(**14408**)	**25201**(**24305**)
Mon. cosine/lin	704(**338**)	2176(**1324**)	4081(**2676**)	10313(9115)	15941(15358)	27143(26561)

Table 5.2: *Results of Parkway and Mondriaan. The Mondriaan results are averages over 5 runs; the best of the 5 runs are stated in brackets. The Parkway results are averages over* 10 *runs. The best averages and the best bests are boldfaced. The distributions for the input and output vectors are the same, and the allowed load imbalance is* $\epsilon = 0.05$.

5.1 Conclusion

We have seen that Par*k*way fine-grained and 2D Mondriaan both give much lower communication volumes than the 1D row decomposition. Moreover, the improvements are of the same order of magnitude. Potentially, the Mondriaan decomposition is much faster than the fine-grained decomposition, and requires much less memory because the hypergraphs involved are roughly a factor 10 smaller. Assuming that the coarsening phase takes is the overhead, this could speed up the partitioning by a factor 100. The column and row scaling is also beneficial in this case, although it is not always an improvement. For better comparison we would need more runs, but we where limited in time to obtain these.

Chapter 6

Conclusions and future work

6.1 Spectral graph theory

Since very little is known about exact solutions of the partitioning problems, it is hard to know how good a given hypergraph partitioner really is. It would be nice to have lower bounds on the communication volume, for example. We have seen that we can derive some lower bounds, using spectral graph theory, but these where lower bounds on the scaled communication volume. It may be possible to formulate the Laplacian of the hypergraph in a different way, to obtain a lower bound on the communication volume.

Minimizing $\mathbf{w}^T \mathcal{L} \mathbf{w}$ minimizes the scaled communication volume. We have seen that we can derive some theoretical justification for the linear row scaling from this. Further investigation could yield more ideas.

6.2 Inner product matching

We have seen that various ways of scaling the inner product do give better results. We did not isolate the effect of the coarsening, so our experiments are not solid evidence that these are indeed the best possible.
In our experiments, the matching order was determined by the weight of the vertices, and not by the degree of the vertices. At first, by error, we also used the weight to scale the inner product. Although the results were practically the same as those of the degree scaled experiments, it is intuitively not 'right' to use the weight for scaling. For a match of two identical vertices, the weight will be twice the degree, and for poor matches the weight will be nearly the vertex degree. This holds for all vertices, so the order should not change too much. We expect therefore, that the order will not change to much and the results will be roughly the same when using the degree to determine the matching order.
Thorough experiments with different matching strategies have also been done in [6]. Most of the strategies

are based on constructing a clique-net graph of the hypergraph, and some of them are actually inner-product matching in disguise. One of these in particular (RHM) is a row-scaled inner product, with $f(\alpha) = \frac{\alpha^2-\alpha}{2}$. The inner product matching, however, is intuitively easier to understand, and deals with the hypergraph directly, without having to construct a clique-net graph.

6.3 The fine-grained hypergraph model

The fine-grained hypergraph model is the most generic formulation of the matrix partitioning problem. In theory, partitioning the fine-grained hypergraph should give the best results, since row or column partitioning is simply fine-grained partitioning with constraints, imposed by the rows or columns. The problem is that it proves difficult for the hypergraph partitioners to deal with this extra freedom. Somehow, there is not enough 'global' information available, and the partitioner gets stuck in a local minimum. Restricting the non-zeros in the rows or columns to stick together is one way of providing such global information. In row or column partitioning these constraints are hard; it is just not possible to take rows or columns apart. Maybe these hard constraints could be turned into soft constraints, by allowing rows or columns to break up if the gain is high enough. Another variant would be to keep rows or columns together during the coarsening phase, and to let them free during the refinement.

6.4 Partitioning PageRank matrices

It has been shown that different 2D hypergraph partitioning methods (fine-grained and Mondriaan) can decrease the communication overhead for the parallel PageRank algorithm tremendously compared to 1D partitioning. Because the row/column hypergraphs are much smaller than the fine-grained hypergraphs, the 2D Mondriaan approach is potentially much faster. The nature of the PageRank algorithm makes it necessary to distribute the input and output vectors equally. Since the matrix is not symmetric, this is an extra constraint on the partitioning. If it were possible to modify the algorithm so that it could efficiently deal with unequally distributed input and output vectors the overhead could be reduced even further.

Bibliography

[1] C.J. Alpert and A.B. Kahng. Geometric embeddings for faster and better multi-way netlist partitioning. In *Design Automation Conference*, pages 743–748, 1993.

[2] R. H. Bisseling and W. Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, 21:47–65, 2005. Special Issue on Combinatorial Scientific Computing.

[3] R.H. Bisseling. *Parallel Scientific Computing. A Structured Approach using BSP and MPI.* Oxford University Press, 2004.

[4] R.H. Bisseling, J. Byrka, S. Cerav-Erbas, N. Gvozdenović, M. Lorenz, R. Pendavingh, C. Reeves, M. Röger, and A. Verhoeven. Partitioning a call graph. In *Proceedings Study Group Mathematics with Industry 2005, Amsterdam.* CWI, Amsterdam, 2005.

[5] M. Bolla. Spectra, euclidean representations and clusterings of hypergraphs. *Discrete Math.*, 117(1-3):19–39, 1993.

[6] E. Boman, K. Devine, R. Heaphy, B. Hendrickson, M. Heroux, and R. Reis. Ldrd report: Parallel repartitioning for optimal solver performance. report SAND2004-0365, Sandia National Laboratories, February 2004.

[7] J.T. Bradley, D.V. de Jager, W.J. Knottenbelt, and A. Trifunovic. Hypergraph partitioning for faster parallel PageRank computation. In *Proc. 2nd European Performance Evaluation Workshop*, pages 155–171, 2005.

[8] T.N. Bui and C. Jones. A heuristic for reducing fill-in in sparse matrix factorization. In *Proc. of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[9] Ü.V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst*, 10(7):673–693, 1999.

[10] Ü.V. Çatalyürek and C. Aykanat. *PaToH: Partitioning Tool for Hypergraphs*, November 1999.

[11] Ü.V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2D decomposition of sparse matrices. In *Proc. 15th International Parallel & Distributed Processing Symposium*, page 118, 2001.

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

[13] M. Fiedler. Algrbraic connectivity of graphs. *Czech. Math. J.*, 23:298–305, 1973.

[14] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czech. Math. J.*, 25:619–633, 1975.

[15] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[16] G.H. Golub and C.F. van Loan. *Matrix Computations.* Johns Hopkins University Press, third edition, 1996.

[17] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix ordering. *IBM J. Res. Develop.*, 41(1/3):171–183, 1997.

[18] S. W. Hadley, B. L. Mark, and A. Vannelli. An efficient eigenvector approach for finding netlist partitions. *IEEE Transactions on Computer-Aided Design*, 11:885–892, 1992.

[19] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? *Lecture Notes in Computer Science*, 1(457):218–225, 1998.

[20] B. Hendrickson and T.G. Kolda. Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing. *SIAM J. Sci. Comput.*, 21(6):2048–2072, November 2000.

[21] B. Hendrickson and R. Leland. The Chaco user's guide, version 2.0. Tech. Rep. SAND94-2692, Sandia National Labs, Albuquerque, NM, October 1994.

[22] W.P. Jones and G. Furnas. Pictures of relevance : A geometric analysis of similarity measures. *Journal of the American Society for Information Science*, 38(6):420–442, 1987.

[23] G. Karypis and V. Kumar. *MeTis: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 3.0.* Univ. of Minnesota, Dept. of Computer Science and Engineering, Army HPC Reseach Center, 1998.

[24] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Journal*, 49:291–307, 1972.

[25] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley, 1990.

[26] N.F. Michelena and Y. Papalambros. A hypergraph framework for optimal model-based decomposition of design problems. *Computational Optimization and Applications*, 8:173–196, 1997.

[27] Z. Drmac M.W. Berry and E.R. Jessup. Matrices, vector spaces and information retrieval. *SIAM Review*, 41(2):335–362, 1999.

[28] M. Mustafa Ozdal and C. Aykanat. Hypergraph models and algorithms for data-pattern-based clustering. *Data Min. Knowl. Discov*, 9(1):29–57, 2004.

[29] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report SIDL-WP-1999-0120, Stanford University, November 1999.

[30] A. Pothen, H.D. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.

[31] A. Trifunovic and W.J. Knottenbelt. Par*k*way 2.0: A parallel multilevel hypergraph partitioning tool. In *ISCIS*, pages 789–800, 2004.

[32] L.G Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[33] B. Vastenhouw and R.H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, March 2005.

[34] K.C.F. Maguire Y.F. Hu and R.J. Blake. A multilevel unsymmetric matrix ordering algorithm for parallel process simulation. *Comput. Chem. Engrg.*, 23:1631–1647, 2000.