Partitioning Sparse Rectangular Matrices for Parallel Processing

Tamara G. Kolda

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6367. kolda@msr.epm.orl.gov.

Abstract. We are interested in partitioning sparse rectangular matrices for parallel processing. The partitioning problem has been well-studied in the square symmetric case, but the rectangular problem has received very little attention. We will formalize the rectangular matrix partitioning problem and discuss several methods for solving it. We will extend the spectral partitioning method for symmetric matrices to the rectangular case and compare this method to three new methods — the alternating partitioning method and two hybrid methods. The hybrid methods will be shown to be best.

1 Introduction

Organizing the nonzero elements of a sparse matrix into a desirable pattern is a key problem in many scientific computing applications, particularly load balancing for parallel computation. In this paper we are interested in ordering the nonzeros of a given matrix into approximate block diagonal form via permutations. This problem corresponds directly to the partitioning problem in graph theory, and so is often referred to as matrix partitioning.

The partitioning problem has been well-studied in the symmetric case [1, 2, 4, 11, 12, 13, 14, 15, 16, 21, 22, 23]. The rectangular partitioning problem, however, has received very little attention; the primary reference in this area is Berry, Hendrickson, and Raghavan [3] on envelope reduction for hypertext matrices.

Let $A$ denote a sparse rectangular $m \times n$ matrix. We will assume throughout that we are working with pattern (0-1) matrices, but the results and methods can easily be extended to nonnegative weighted matrices. Our goal is to partition $A$ into a block $2 \times 2$ matrix so that most of the nonzeros are on the block diagonal and so that each block diagonal has about the same number of nonzeros. In other words, we wish to find permutation matrices $P$ and $Q$ such that

$$B \equiv PAQ = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix},$$

* This work was supported by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy, under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation.
DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.
where \( B_{12} \) and \( B_{21} \) are as sparse as possible and the block rows or block columns each have about the same number of nonzeros. In order to avoid a trivial solution (e.g., \( B_{11} = A \)), we require that \( B_{11} \) have \( p \) rows and \( q \) columns where \( p \) is some integer between 1 and \( m - 1 \) and \( q \) is some integer between 1 and \( n - 1 \). The values \( p \) and \( q \) may or may not have been chosen in advance; typically we will want \( p \approx m/2 \) and \( q \approx n/2 \) to maintain load balance. If there exists \( P \) and \( Q \) such that \( B_{12} \) and \( B_{21} \) are identically zero, then we say that \( A \) is block diagonalizable. If we wish to partition \( A \) in a block \( 2^k \times 2^k \) matrix, we can recursively partition the block diagonals.

The matrix partitioning problem is equivalent to the edge-weighted graph partitioning problem: Given an undirected edge-weighted graph, partition the nodes into two sets of given sizes such that the sum of the weights of the edges that pass between the two sets is minimized. The graph partitioning problem is a well-known NP-complete problem (see problem ND14 on p. 209 in Garey and Johnson [6]). A rectangular \( m \times n \) matrix corresponds to a bipartite graph [3] with \( m \) left nodes and \( n \) right nodes. There is an edge between left node \( i \) and right node \( j \) if \( a_{ij} \) is nonzero, and the weight of the edge is one. See Fig. 1 for an illustration. Suppose that we partition \( A \) so that the union of the first \( p \) rows and the first \( q \) columns form one partition and the remaining rows and columns form the other partition. The edges passing between the two partitions correspond to the nonzeros in the off-diagonal blocks of the partitioned matrix.

\[
\begin{bmatrix}
1 & 2 & 3 \\
1 & x & \\
2 & x & \\
5 & x & x \\
4 & x & \\
5 & x & \\
\end{bmatrix}
\]

Fig. 1. The bipartite graph of a rectangular matrix.

Observe that the graph of \( A \) is disconnected if and only if the matrix \( A \) is block diagonalizable. Throughout we will assume that the graph of \( A \) is connected. If it is not, we will re-order the matrix so that it is block diagonalized with the blocks in decreasing order of size. We will only work with components that cross the boundary of the desired partition. In the discussion of the theory, we will assume that the graph of \( A \) is connected.

Many iterative methods, e.g., LSQR[20] require matrix-vector and matrix-vector-transpose multiplies with rectangular matrices. In Sect. 2 we will describe how to implement these kernels to take advantage of the partitioned matrix.

In Sect. 3, we will present several algorithms for the rectangular partitioning problem. We will discuss the well-known spectral partitioning method and show
how it can be applied to this problem. We will also introduce a new alternating partitioning method as well as two hybrid strategies.

In Sect. 4, we will compare the various partitioning methods, and show that the hybrid methods are the best.

2 Parallel Matrix-Vector Multiplication

We propose the following parallel implementations for the matrix-vector and matrix-transpose-vector multiplications. Suppose that we have \( p = 2^k \) processors. We partition \( A \) into a block \( r \times r \) matrix,

\[
A = \begin{bmatrix}
  A_{11} & A_{12} & \cdots & A_{1r} \\
  A_{21} & A_{22} & \cdots & A_{2r} \\
  \vdots & \vdots & \ddots & \vdots \\
  A_{r1} & A_{r2} & \cdots & A_{rr}
\end{bmatrix},
\]

so that most of the nonzeros are in the diagonal blocks. Here block \((i,j)\) is of size \( m_i \times n_j \) where \( \sum_i m_i = m \) and \( \sum_j n_j = n \).

**Matrix-Vector Multiply (Block Row).** We do the following on each processor to compute \( y = Ax \):

1. Let \( i \) denote the processor id. This processor owns the \( i \)th block row of \( A \), that is, \([ A_{1i}, A_{2i}, \ldots, A_{ri} \] \), and \( x_i \), the \( i \)th block of \( x \) of length \( n_i \).
2. Send a message to each processor \( j \neq i \) for which \( A_{ij} \neq 0 \). This message contains only those elements of \( x_j \) corresponding to nonzero columns in \( A_{ij} \).
3. While waiting to receive messages, the processor computes the contribution from the diagonal matrix block, \( y_i^{(i)} = A_{ii} x_i \). The block \( A_{ii} \), while still sparse, may be dense enough to improve data locality.
4. Then, for each \( j \neq i \) such that \( A_{ij} \) is nonzero, a message is received containing a sparse vector \( \tilde{x}_j \) that only has the elements of \( x_j \) corresponding to nonzero columns in \( A_{ij} \), and \( y_j^{(i)} = A_{ij} \tilde{x}_j \), is computed. (We assume that processor \( i \) already knows which elements to expect from processor \( j \).)
5. Finally, the \( i \)th block of the product \( y \) is computed via the sum \( y_i = \sum_j y_j^{(i)} \).

Block \( y_i \) is of size \( m_i \).

**Matrix-Transpose-Vector Multiply (Block Row).** To compute \( z = A^T v \), each processor does the following:

1. Let \( i \) denote the processor id. This processor owns \( v_i \), the \( i \)th block of \( v \) of size \( m_i \), and the \( i \)th block row of \( A \).
2. Compute \( z_j^{(i)} = A_{ij} v_i \), for each \( j \neq i \) for which \( A_{ij} \neq 0 \). Observe that the number of nonzeros in \( z_j^{(i)} \) is equal to the number of nonzero rows in \( A_{ij}^T \), i.e., the number of nonzero columns in \( A_{ij} \). Send the nonzero\(^1\) elements of \( z_j^{(i)} \) to processor \( j \).

---

\(^1\) Here we mean any elements that are guaranteed to be zero by the structure of \( A_{ij} \). Elements that are zero by cancellation are still communicated.
3. While waiting to receive messages from the other processors, compute the diagonal block contribution $z_i^{(t)} = A_{ii}^T v_i$.

4. From each processor $j$ such that $A_{ji} \neq 0$, receive $z_j^{(j)}$ which contains only the nonzero elements of $z_i^{(j)}$. (Again, we assume that processor $i$ already knows which elements to expect from processor $j$.)

5. Compute the $i$th component of the product, $z_i = z_i^{(t)} + \sum_{j \neq i} z_j^{(j)}$. Block $z_i$ is of size $n_i$.

Block column algorithms are analogous to those given for the block row layout. Observe that sparse off-diagonal blocks result in less message volume. See Hendrickson and Kolda [9] for more detail on the algorithm and for more details on potential applications.

3 Algorithms for the Rectangular Partitioning Problem

Here we will discuss how the well-known spectral method can be applied to the rectangular problem and introduce a new method that can be used on its own or in combination with other methods. The spectral method will be used as a basis for comparison to the new methods in Sect. 4.

3.1 Spectral Partitioning

In the symmetric problem, spectral partitioning based on the Fiedler vector is a well-known technique; see, for example, Pothen, Simon, and Liou [21]. Many people have studied the effectiveness of spectral graph partitioning; for example, Guattery and Miller [8] show that spectral partitioning can be bad, while Spielman and Teng [23] show that it can be good.

One natural way to approach the rectangular problem is to symmetrize the matrix $A$, yielding the $(m+n) \times (m+n)$ matrix and apply spectral partitioning to the symmetrized matrix. This approach is used by Berry et al. [3]. Note that the graphs of $\tilde{A}$ and $A$ are the same.

In order to apply spectral partitioning, we compute the Laplacian of $\tilde{A}$,

$$L = D - \tilde{A},$$

where $D = \text{diag}(d_1, d_2, \ldots, d_{m+n})$ and $d_i = \sum_j \tilde{a}_{ij}$. The matrix $L$ is symmetric and semi-positive definite; furthermore, the multiplicity of the zero eigenvalue must be one since we are assuming that the graph of $A$, and hence of $\tilde{A}$, is connected [5]. Let $w$ denote the Fiedler vector of $L$, that is, the eigenvector corresponding to the smallest positive eigenvalue of $L$. Let $u$ denote the first $m$ and $v$ the last $n$ elements of $w$, and sort the elements of $u$ and $v$ so that

$$u_1 \geq u_2 \geq \cdots \geq u_m,$$
$$v_1 \geq v_2 \geq \cdots \geq v_n.$$
Then \{i_1, i_2, \ldots, i_m\} and \{j_1, j_2, \ldots, j_n\} define the row and column partitions respectively. In other words, assign rows \(i_1, i_2, \ldots, i_p\) and columns \(j_1, j_2, \ldots, j_q\) to the first partition and the remaining rows and columns to the second partition. Note that the ordering is independent of \(p\) and \(q\). This means that the values of \(p\) and \(q\) may be fixed in advance as something like \([m/2]\) and \([n/2]\) respectively, or they may be chosen after the ordering has been computed to ensure good load balancing.

In Sect. 4 we will use this method as a basis for comparison for our new methods.

### 3.2 The Alternating Partitioning Method

Rather than trying to compute both the row and column partitions simultaneously as is done in the spectral method, the new method proposed in this section focuses on one partition at a time, switching back and forth. This method is derived from the Semi-Discrete Matrix Decomposition, a decomposition that has been used for image compression [19] and information retrieval [17, 18].

Before we describe the method, we will re-examine the problem. If we let \(I\) denote the set of row indices that are permuted to a value less than or equal to \(p\) and correspondingly let \(I^c\) denote the set of row indices permuted to a value greater than \(p\) and define the set \(J\) in an analogous way for the columns, then we can write the rectangular partitioning problem as a maximization problem,

\[
\max_{I,J} \sum_{i \in I} \sum_{j \in J} a_{ij} + \sum_{i \in I^c} \sum_{j \in J} a_{ij} - \sum_{i \in I} \sum_{j \in J} a_{ij} ,
\]

\[
\text{s.t. } I \subseteq \{1, 2, \ldots, m\} , J \subseteq \{1, 2, \ldots, n\} , |I| = p, |J| = q .
\]

Here the objective function is the sum of the nonzeros on the block diagonal minus the sum of the elements off the block diagonal.

We can then rewrite this problem as an integer programming problem. Let \(x\) be a vector that defines the set membership for each row index; that is, \(x_i = 1\) if row \(i\) is in \(I\), and \(x_i = -1\) if row \(i\) is in \(I^c\), and let the vector \(y\) be defined in an analogous way for the columns. Then we can rewrite problem (1) as

\[
\max x^T A y ,
\]

\[
\text{s.t. } x_i = \pm 1 , y_j = \pm 1 , x^T e = 2p - m , y^T e = 2q - n ,
\]

where \(e\) denotes the ones vector whose length is implied by the context.

Although we cannot solve (2) exactly, we can use an alternating method to get an approximate solution. We fix the partition for, say, the right nodes \((y)\), and then compute the best possible partition for the left nodes \((x)\). Conversely, we then fix the partition for the left nodes, and compute the best possible partition for the right nodes, and so on.
Suppose that we have fixed the partition for the right nodes. To determine the best partition of the left nodes, we need to solve

\[
\begin{align*}
\max x^T s, \\
\text{s.t. } x_1 &= \pm 1, \\
x^T e &= 2p - m,
\end{align*}
\]

where \( s = A y \) is fixed. The solution to this problem can be computed exactly. If we sort the entries of \( s \) so that

\[
s_{i_1} \geq s_{i_2} \geq \cdots \geq s_{i_m},
\]

then \( x \) defined by \( x_{i_1} = x_{i_2} = \cdots = x_{i_p} = +1 \) and \( x_{i_p+1} = x_{i_p+2} = \cdots = x_{i_m} = -1 \) is the exact solution to (3). Observe that the ordering of the elements of \( s \) does not depend on the value for \( p \). If \( p \) has not been specified ahead of time, we would choose \( p \) to ensure load balancing. However, note that then \( p \) may be changing every iteration. An analogous procedure would be employed to find \( y \) when \( x \) is fixed.

Assuming \( p \) and \( q \) are fixed, each time we fix one side's partition and then compute the other, we are guaranteed that the value of the objective will never decrease. In other words, let \( x^{(k)} \) and \( y^{(k)} \) denote the partitions at the \( k \)th iteration of the method, and let \( f_k \) denote the objective value, \( x^{(k)}^T A y^{(k)} \); then \( f_{k+1} \geq f_k \) for all \( k \). In the experiments presented in this paper, the method terminates when the objective value stops increasing. Alternatively, the method could terminate after at most some fixed number of iterations.

This method is called the alternating partitioning (AP) method and is specific to the rectangular problem since we are dealing with both row and column partitions and so we can alternate between working with one and then the other. In the symmetric case, we are only dealing with one partition.

We have not yet specified how to choose the first partition when we start the iterations, but that choice is important. In the standard method, we simply use the identity partition; however the next subsection will present two hybrid methods that use other techniques to generate a starting partition.

### 3.3 Hybrid Alternating Partitioning Methods

Since the alternating partitioning method is a greedy method, its key to success is having a good starting partition. Here we propose two possibilities.

**Hybrid Spectral – Alternating Partitioning Method.** This method uses the partition generated by the spectral method described in Sect. 3.1 as the starting partition for the alternating partitioning method.

**Hybrid RCM – Alternating Partitioning Method.** The Reverse Cuthill-McKee (RCM) method is not generally used as a partitioning method, but it generates a good inexpensive starting partition for the alternating partitioning method.
RCM is typically used for envelope reduction on symmetric matrices and is based on the graph of the matrix. Essentially, the method chooses a starting node and labels it 1. It then consecutively labels the nodes adjacent to it, then labels the nodes adjacent to them, and so forth. Once all the nodes are labelled, the ordering is reversed (hence the name). See George and Liu [7] for further discussion. In the nonsquare or nonsymmetric case, we apply RCM to the symmetrized matrix as was done by Berry et al. [3].

4 Experimental Results

In this section we compare the various partitioning methods presented in Sect. 3 on a collection of matrices listed in Table 1. These matrices were obtained from Matrix Market with the exception of ccealink, man1, man2, and nhse400 which were provided by Michael Berry and are those used in Berry et al. [3]. These matrices range in size from 100 × 100 to 4000 × 400. All of the square matrices are structurally nonsymmetric.

Table 1. Rectangular test matrices.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Rows</th>
<th>Columns</th>
<th>Nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>bf782a</td>
<td>782</td>
<td>782</td>
<td>7514</td>
</tr>
<tr>
<td>ccealink</td>
<td>1778</td>
<td>850</td>
<td>2388</td>
</tr>
<tr>
<td>gre_115</td>
<td>115</td>
<td>115</td>
<td>421</td>
</tr>
<tr>
<td>11c1033</td>
<td>1033</td>
<td>320</td>
<td>4732</td>
</tr>
<tr>
<td>impcol_a</td>
<td>207</td>
<td>207</td>
<td>572</td>
</tr>
<tr>
<td>impcol_c</td>
<td>137</td>
<td>137</td>
<td>411</td>
</tr>
<tr>
<td>impcol_d</td>
<td>425</td>
<td>425</td>
<td>1339</td>
</tr>
<tr>
<td>impcol_e</td>
<td>225</td>
<td>225</td>
<td>1308</td>
</tr>
<tr>
<td>man1</td>
<td>1853</td>
<td>625</td>
<td>3706</td>
</tr>
<tr>
<td>man2</td>
<td>1426</td>
<td>850</td>
<td>2388</td>
</tr>
<tr>
<td>nhse400</td>
<td>4233</td>
<td>400</td>
<td>5119</td>
</tr>
<tr>
<td>nnc261</td>
<td>261</td>
<td>261</td>
<td>1500</td>
</tr>
<tr>
<td>watson3</td>
<td>124</td>
<td>124</td>
<td>780</td>
</tr>
<tr>
<td>west0132</td>
<td>132</td>
<td>132</td>
<td>414</td>
</tr>
<tr>
<td>west0156</td>
<td>156</td>
<td>156</td>
<td>371</td>
</tr>
<tr>
<td>west0479</td>
<td>479</td>
<td>479</td>
<td>1910</td>
</tr>
<tr>
<td>utm300</td>
<td>300</td>
<td>300</td>
<td>3155</td>
</tr>
</tbody>
</table>

Tables 2, 3, and 4 show the results of partitioning the rectangular matrices into block 2×2, 8×8, and 16×16 matrices. In these experiments, all the matrices were converted to pattern (0-1) matrices, but we could have converted them to nonnegatively weighted matrices instead. When partitioning into a block 2 × 2

http://math.nist.gov/MatrixMarket/
matrix, we choose \( p = \lfloor m/2 \rfloor \) and \( q = \lfloor n/2 \rfloor \). (Alternatively, we could compute \( p \) and \( q \) on the fly.) When partitioning into more blocks, we first partition into a block \( 2 \times 2 \) matrix and recursively partition the diagonal blocks into block \( 2 \times 2 \) matrices until the desired number of blocks is reached.

The tables are formatted as follows. Each row corresponds to a given matrix whose name is specified in the first column. The second column lists the number of nonzeros outside of the block diagonal in the original matrix. The number of nonzeros outside of the block diagonal is equivalent to the number of edge cuts in the graph partitioning problem. Columns 3–6 list the number of nonzeros outside the block diagonal after applying the method listed in the column header to the original matrix. The number in parentheses is the time in seconds it took to compute the ordering. All timings were done in MATLAB.

Spectral partitioning requires the second eigenvector corresponding to the second smallest eigenvalue, and this was computed using the MATLAB EIGS routine. (Note that EIGS has a random element to it, so the results for the spectral and hybrid spectral–alternating partitioning method cannot be repeated exactly.) The alternating partitioning method was implemented using our own code. The MATLAB SYMRCM routine was used to compute the RCM ordering.

Table 2. Approximate block diagonalization of rectangular matrices into block \( 2 \times 2 \) matrices.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Nonzeros outside of Block ( 2 \times 2 ) Diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orig</td>
<td>Spectral</td>
</tr>
<tr>
<td>bwf782a</td>
<td>2438</td>
</tr>
<tr>
<td>ccealink</td>
<td>324</td>
</tr>
<tr>
<td>cre_115</td>
<td>104</td>
</tr>
<tr>
<td>illc1033</td>
<td>2336</td>
</tr>
<tr>
<td>impcol_a</td>
<td>59</td>
</tr>
<tr>
<td>impcol_c</td>
<td>46</td>
</tr>
<tr>
<td>impcol_d</td>
<td>48</td>
</tr>
<tr>
<td>impcol_e</td>
<td>152</td>
</tr>
<tr>
<td>man1</td>
<td>803</td>
</tr>
<tr>
<td>man2</td>
<td>759</td>
</tr>
<tr>
<td>n3e400</td>
<td>213</td>
</tr>
<tr>
<td>ncc261</td>
<td>75</td>
</tr>
<tr>
<td>watson3</td>
<td>157</td>
</tr>
<tr>
<td>west0132</td>
<td>87</td>
</tr>
<tr>
<td>west0156</td>
<td>231</td>
</tr>
<tr>
<td>west0479</td>
<td>755</td>
</tr>
<tr>
<td>utm300</td>
<td>266</td>
</tr>
</tbody>
</table>

We are using the spectral method as a basis for comparison for our new methods. Let us first consider the alternating partitioning method. In the \( 2 \times 2 \) and \( 8 \times 8 \) tests, the spectral method does better than the alternating partitioning
Table 3. Approximate block diagonalization of rectangular matrices into block $8 \times 8$ matrices.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Orig</th>
<th>Spectral</th>
<th>AP</th>
<th>RCM-AP</th>
<th>Spec-AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfi782a</td>
<td>3872</td>
<td>832 (55.35)</td>
<td>1236 (5.02)</td>
<td>963 (5.72)</td>
<td>804 (55.03)</td>
</tr>
<tr>
<td>cceilink</td>
<td>684</td>
<td>991 (36.90)</td>
<td>813 (5.61)</td>
<td>286 (6.37)</td>
<td>216 (32.05)</td>
</tr>
<tr>
<td>gre_115</td>
<td>265</td>
<td>131 (3.92)</td>
<td>146 (0.55)</td>
<td>128 (0.64)</td>
<td>118 (4.26)</td>
</tr>
<tr>
<td>i11c1033</td>
<td>4066</td>
<td>2474 (31.35)</td>
<td>1640 (3.17)</td>
<td>1398 (3.83)</td>
<td>1399 (32.72)</td>
</tr>
<tr>
<td>impcol_a</td>
<td>269</td>
<td>56 (6.66)</td>
<td>67 (0.73)</td>
<td>83 (0.88)</td>
<td>48 (5.70)</td>
</tr>
<tr>
<td>impcol_c</td>
<td>170</td>
<td>108 (4.19)</td>
<td>106 (0.57)</td>
<td>94 (0.66)</td>
<td>85 (4.14)</td>
</tr>
<tr>
<td>impcol_d</td>
<td>260</td>
<td>171 (15.02)</td>
<td>167 (1.37)</td>
<td>163 (1.82)</td>
<td>128 (15.43)</td>
</tr>
<tr>
<td>impcol_e</td>
<td>724</td>
<td>303 (6.70)</td>
<td>173 (0.91)</td>
<td>153 (1.10)</td>
<td>157 (6.81)</td>
</tr>
<tr>
<td>man1</td>
<td>1492</td>
<td>518 (82.38)</td>
<td>1249 (4.91)</td>
<td>578 (6.49)</td>
<td>433 (83.09)</td>
</tr>
<tr>
<td>man2</td>
<td>2207</td>
<td>449 (59.49)</td>
<td>904 (3.90)</td>
<td>524 (4.95)</td>
<td>326 (60.79)</td>
</tr>
<tr>
<td>nhse400</td>
<td>1001</td>
<td>1493 (143.98)</td>
<td>1142 (7.99)</td>
<td>227 (10.25)</td>
<td>234 (148.47)</td>
</tr>
<tr>
<td>nnc261</td>
<td>596</td>
<td>339 (9.23)</td>
<td>398 (1.04)</td>
<td>353 (1.27)</td>
<td>341 (9.99)</td>
</tr>
<tr>
<td>watson3</td>
<td>350</td>
<td>285 (4.02)</td>
<td>236 (0.56)</td>
<td>234 (0.71)</td>
<td>232 (4.40)</td>
</tr>
<tr>
<td>west0132</td>
<td>308</td>
<td>97 (4.01)</td>
<td>111 (0.57)</td>
<td>104 (0.67)</td>
<td>82 (4.14)</td>
</tr>
<tr>
<td>west0156</td>
<td>337</td>
<td>63 (4.13)</td>
<td>73 (0.59)</td>
<td>53 (0.76)</td>
<td>42 (4.17)</td>
</tr>
<tr>
<td>west0479</td>
<td>1309</td>
<td>427 (18.58)</td>
<td>420 (1.65)</td>
<td>413 (2.08)</td>
<td>274 (18.75)</td>
</tr>
<tr>
<td>utm300</td>
<td>1359</td>
<td>991 (12.19)</td>
<td>1211 (1.59)</td>
<td>830 (1.98)</td>
<td>888 (12.96)</td>
</tr>
</tbody>
</table>

Table 4. Approximate block diagonalization of rectangular matrices into block $16 \times 16$ matrices.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Orig</th>
<th>Spectral</th>
<th>AP</th>
<th>RCM-AP</th>
<th>Spec-AP</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfi782a</td>
<td>4320</td>
<td>1387 (61.83)</td>
<td>1951 (6.28)</td>
<td>1429 (7.17)</td>
<td>1361 (62.18)</td>
</tr>
<tr>
<td>cceilink</td>
<td>904</td>
<td>1195 (38.56)</td>
<td>1053 (7.27)</td>
<td>358 (7.96)</td>
<td>266 (32.63)</td>
</tr>
<tr>
<td>gre_115</td>
<td>291</td>
<td>171 (6.45)</td>
<td>195 (0.92)</td>
<td>162 (1.06)</td>
<td>153 (6.35)</td>
</tr>
<tr>
<td>i11c1033</td>
<td>4376</td>
<td>2776 (33.45)</td>
<td>2194 (4.12)</td>
<td>1985 (4.92)</td>
<td>1920 (37.83)</td>
</tr>
<tr>
<td>impcol_a</td>
<td>413</td>
<td>125 (8.49)</td>
<td>115 (1.18)</td>
<td>127 (1.46)</td>
<td>100 (8.45)</td>
</tr>
<tr>
<td>impcol_c</td>
<td>252</td>
<td>160 (8.15)</td>
<td>157 (1.04)</td>
<td>143 (1.16)</td>
<td>135 (7.26)</td>
</tr>
<tr>
<td>impcol_d</td>
<td>444</td>
<td>289 (18.86)</td>
<td>278 (1.96)</td>
<td>305 (2.50)</td>
<td>243 (19.83)</td>
</tr>
<tr>
<td>impcol_e</td>
<td>976</td>
<td>615 (9.95)</td>
<td>474 (1.55)</td>
<td>471 (1.86)</td>
<td>450 (9.84)</td>
</tr>
<tr>
<td>man1</td>
<td>1837</td>
<td>796 (91.35)</td>
<td>1479 (6.21)</td>
<td>699 (7.98)</td>
<td>602 (91.92)</td>
</tr>
<tr>
<td>man2</td>
<td>2519</td>
<td>596 (66.93)</td>
<td>1086 (5.06)</td>
<td>671 (6.27)</td>
<td>456 (69.61)</td>
</tr>
<tr>
<td>nhse400</td>
<td>1594</td>
<td>1751 (161.72)</td>
<td>1795 (10.33)</td>
<td>331 (12.64)</td>
<td>428 (165.84)</td>
</tr>
<tr>
<td>nnc261</td>
<td>890</td>
<td>556 (12.69)</td>
<td>558 (1.63)</td>
<td>538 (1.87)</td>
<td>528 (13.08)</td>
</tr>
<tr>
<td>watson3</td>
<td>415</td>
<td>367 (6.39)</td>
<td>289 (0.99)</td>
<td>297 (1.18)</td>
<td>300 (6.90)</td>
</tr>
<tr>
<td>west0132</td>
<td>376</td>
<td>162 (6.34)</td>
<td>152 (0.97)</td>
<td>141 (1.24)</td>
<td>132 (6.88)</td>
</tr>
<tr>
<td>west0156</td>
<td>351</td>
<td>92 (8.11)</td>
<td>87 (1.04)</td>
<td>74 (1.23)</td>
<td>70 (6.43)</td>
</tr>
<tr>
<td>west0479</td>
<td>1618</td>
<td>640 (22.55)</td>
<td>541 (2.33)</td>
<td>477 (2.84)</td>
<td>414 (22.62)</td>
</tr>
<tr>
<td>utm300</td>
<td>1584</td>
<td>1504 (15.40)</td>
<td>1401 (2.21)</td>
<td>1268 (2.70)</td>
<td>1268 (15.95)</td>
</tr>
</tbody>
</table>
method on the majority of matrices, but on the $16 \times 16$ tests, the alternating partitioning method does better than spectral partitioning 11 out of 17 times. In terms of time, the alternating partitioning method is approximately 10 times faster than the spectral method.

The hybrid RCM-AP method is overall better than the alternating partitioning method. It outperforms the spectral method in the majority of matrices in the $8 \times 8$ and $16 \times 16$ tests. This method is the best overall an average of 2.3 times for each block size. The time for the hybrid RCM-AP method is only slightly more than that for the alternating partitioning method, and still about 10 times less than that for the spectral method.

For all three block sizes, we see that the hybrid spectral-AP method is the best overall in terms of reducing the number of nonzeros outside the block diagonal. This method is only slightly more expensive than the spectral method in terms of time and yields consistently better results. Sometimes the improvement is remarkable; see, for example, ccsalink in all four tables. Unfortunately, the spectral and hybrid spectral-AP methods are very expensive in terms of time since they require the computation of some eigenpairs.

Figure 2 shows the effect of different partitioning strategies on the west0156 matrix in the block $8 \times 8$ case.

![Original Matrix](image1)
![Spectral Partitioning](image2)
![Alternating Partitioning](image3)

![Hybrid RCM-AP](image4)
![Hybrid Spectral-AP](image5)

Fig. 2. Comparison of block $8 \times 8$ partitions on west0156.
Given these results, we recommend the hybrid spectral-AP method when quality is the top concern and the hybrid RCM-AP method when both time and quality matter.

5 Conclusions

In this work we introduced the rectangular matrix partitioning problem which is an extension of the symmetric matrix partitioning problem. The rectangular partitioning problem has a number of potential uses in iterative methods such as LSQR.

We introduced the alternating partitioning method as well as two hybrid methods and compared them with a rectangular version of the spectral partitioning method. The hybrid methods compared extremely favorably, and we recommend these as the methods of choice. The hybrid spectral-AP method is the overall best when partition quality is more important than the time to compute the partition, and the hybrid RCM-AP method is recommended when time is more important.

In other work [9, 10], this author and Bruce Hendrickson explore the multilevel method which is known to work very well for the symmetric partitioning problem [2, 11, 12, 13, 15]. We develop a multilevel method specific for bipartite graphs with various refinement strategies including the alternating partitioning method and a version of Kernighan-Lin for bipartite graphs. Eventually, we would also like to examine handling four or eight diagonal blocks directly [1, 11].

Acknowledgments

The author is indebted to Bruce Hendrickson and Dianne O’Leary for many helpful discussions. The author also thanks Eduardo D’Azevedo, Chuck Romine, and the anonymous referees for their reviews, and Mike Berry for providing data.

References


