Partitioning Mathematical Programs for Parallel Solution ⋆

Michael C. Ferris and Jeffrey D. Horn [1]

Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

This paper describes heuristics for partitioning a general $M \times N$ matrix into arrowhead form. Such heuristics are useful for decomposing large, constrained, optimization problems into forms that are amenable to parallel processing. The heuristics presented can be easily implemented using publicly available graph partitioning algorithms. The application of such techniques for solving large linear programs is described. Extensive computational results on the effectiveness of our partitioning procedures and their usefulness for parallel optimization are presented.

Key words: Graph Partitioning, Linear Programming, Bundle Method, Parallel Optimization

1 Introduction

This paper describes several heuristics for partitioning a general $M \times N$ matrix into arrowhead form. Such partitioning is useful in many areas of numerical analysis where several partitioning heuristics exist for the special case of $N \times N$ symmetric matrices [4,9]. We make use of several recent innovations in graph partitioning heuristics to decompose large, constrained optimization problems into forms amenable to parallel processing. This is done by partitioning the large sets of constraints arising in optimization problems into a manageable number of independent blocks of constraints, linked together by relatively few linking variables and coupling constraints.

⋆ This material is based on research supported by National Science Foundation Grants CCR-9157632 and CDA-9024618, the Air Force Office of Scientific Research Grant F49620-94-1-0036 and the AT&T Foundation.

[1] email: {ferris,horn}@cs.wisc.edu

First, the arrowhead form is described and basic results on the correspondence between an $M \times N$ matrix and its associated graph are presented. This correspondence is then used to present heuristics for partitioning a matrix by partitioning the associated graph. The results of the heuristics can be improved to some extent by adding dummy nodes to the associated graph. These dummy nodes enable the resulting blocks to have uneven sizes, and even some of the blocks to be empty. By adding enough dummy nodes to the graph, we are able to accommodate problems that naturally split into fewer than the requested number of blocks.

In Section 3, we present some computational results to demonstrate the effectiveness of our heuristics. We give two sets of results to show how well our partitioning algorithms perform *as partitioning algorithms*, that is, how close do they come to producing a matrix in arrowhead form with the desired number of blocks. In the first set of results, we show the effect of changing the number of dummy nodes in the problem for the complete set of problems in the NETLIB test suite. We detail the percentages of coupling constraints and linking variables, and the ratio of the largest block size to the average, in addition to an overall measure that indicates how well our heuristic performs. This analysis is used to fix the percentage of dummy nodes for the remainder of our computation. In the second set of results we show how well our heuristics performs by taking a problem that is naturally in arrowhead form and randomly permuting its rows and columns. Our heuristics effectively reconstruct an arrowhead form. These results are useful in determining what classes of problems are amenable to this kind of partitioning, what are the relative costs of treating the linking variables and constraints, and how balanced the computational load will be for the parallel processors.

The remainder of the paper shows one way to use the partitioning algorithm for the solution of linear programs. The linking variables are removed and replaced with coupling constraints to which a dual method is applied. The dual problem is a non-smooth optimization problem which is solved by an application of the bundle-level method. Using this approach, we illustrate the utility of partitioning matrices by decomposing the constraint sets of several NETLIB linear programs into a reasonable number of independent constraint blocks and a relatively small number of linking variables and coupling constraints. The resulting problem may be solved in parallel on as many processors as there are independent constraint blocks. The computational results given in Section 4.2 measure the *utility* of our partitioning algorithms for the efficient solution of large-scale, linear programs.

The analysis of this paper does not rely on the linearity of the constraints. Nonlinear programs can use the same technique to exploit underlying structure in the constraint set and enable the efficient solution of such problems using decomposition techniques such as those found in [5,6,29]. Furthermore, although many modern modeling languages and systems allow block structure to be specified during problem formulation, the techniques we outline here can be used to modify such a partition to take full advantage of the number and relative performance of the available parallel processing units.

2 Matrix Partitioning Algorithms

Definition 1 *A matrix is said to be in arrowhead form if it has the following structure:*

$$\begin{pmatrix} B_1 & & & & C_1 \\ & B_2 & & & C_2 \\ & & \ddots & & \vdots \\ & & & B_K & C_K \\ R_1 & R_2 & \dots & R_K & D \end{pmatrix}.$$

Here $B_i \in \mathbb{R}^{m_i \times n_i}$, $C_i \in \mathbb{R}^{m_i \times p}$, $R_i \in \mathbb{R}^{q \times n_i}$ and $D \in \mathbb{R}^{q \times p}$. We call each B_i a *block* and note that in the matrix above there are K such blocks. We let $M := \sum_{i=1}^{K} m_i + q$ and $N := \sum_{j=1}^{K} n_j + p$ be the row and column dimensions of the matrix respectively.

Definition 2 *We call each row of the $q \times N$ submatrix*

$$\begin{pmatrix} R_1 & R_2 & \dots & R_K & D \end{pmatrix}$$

a coupling constraint*.*

In general, these rows link together or restrict the column spaces of blocks, resulting in the column space for the entire matrix. Such a row may restrict the column space of one block B_i based on the column space of another block B_j. In this event, the blocks B_i and B_j are said to be *linked* or *coupled* by such a row. The reader should note that *coupling constraints* appear as *rows* in the arrowhead form of the matrix.

Definition 3 *We call each column of the submatrix*

$$\begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_K \\ D \end{pmatrix}$$

a linking column*.*

In general, these columns link together or restrict the row spaces of blocks, resulting in the row space for the entire matrix. Such a column may restrict the row space of one block B_i based on the row space of another block B_j. In this event, the blocks B_i and B_j are also said to be *linked* by such a column.

We note that p and q may take the value 0, in which case either the linking columns or the coupling constraints will be missing. If $p = 0, q \neq 0$ or $p \neq 0, q = 0$, the resulting matrix is called a *singly-bordered*, block-diagonal matrix. If both p and q are equal to 0, then we will simply call the matrix block-diagonal.

We will later give a procedure whereby all of the linking columns can be removed by adding some columns to various blocks and extra coupling constraints, thus transforming an arrowhead form into a singly-bordered block-diagonal form.

An important concept in what follows is that of the associated graph of a matrix [10].

Definition 4 *Given a matrix $A_{M\times N}$, the associated graph of A, denoted by $G(A)$ is the pair (V, E) satisfying:*

(i) $V = R \cup C$, $R = \{r_1, r_2, \ldots, r_M\}$, $C = \{c_1, c_2, \ldots, c_N\}$.
(ii) $(r_i, c_j) \in E$ *if* $r_i \in R$, $c_j \in C$, *and* $a_{i,j} \neq 0$.

Note that the $G(A)$ is a bipartite graph, with (R, C) being a bipartition. That is, there are no edges joining elements of R to R, or C to C. The set R is the set of *row vertices* of $G(A)$ and C is the set of *column vertices* of $G(A)$.

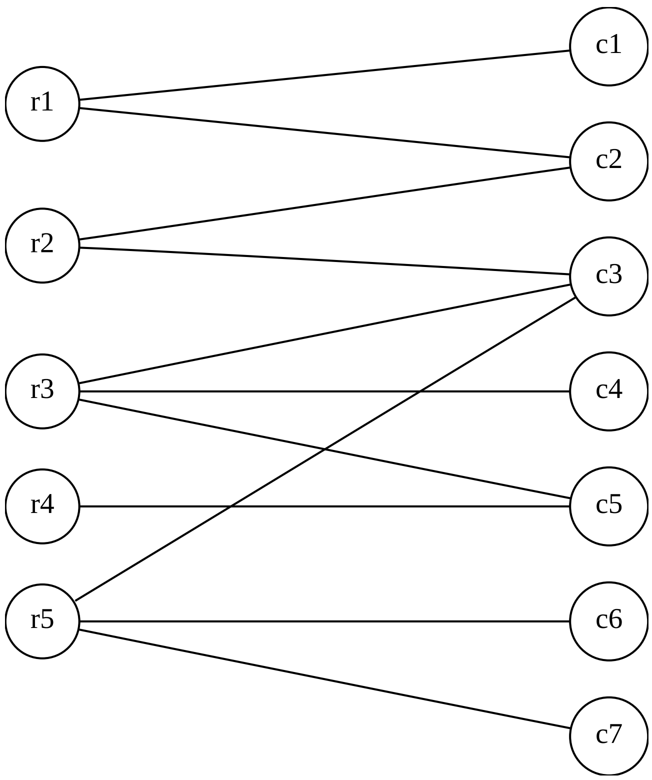

Fig. 1. The associated bipartite graph G(A)

For example, given the following 5×7 matrix

$$A = \begin{pmatrix} x & x & & & & & \\ & x & x & & & & \\ & & x & x & x & & \\ & & & & x & & \\ & & x & & & x & x \end{pmatrix}$$

where x denotes a non-zero entry, we have $G(A)$ given in Figure 1.

The following definition is key to the algorithm that we use to create the matrix in arrowhead form. It relates to a general graph; in our work, we use it for the associated graph of a matrix.

Definition 5 *Given a graph $G = (V, E)$, and an integer K, a* partition *of G is a partition of the set V of vertices of G into K subsets. The* cost *of such a partition is the number of edges in E that connect vertices in* different *subsets of the partition of V.*

The general technique the we use to partition the graph is a multilevel procedure. At each level, clusters of nodes with high connectivity are merged to form a supernode, reducing in the size of the graph. Levels are iteratively formed until the condensed graph is suitably small.

A spectral method is applied to the condensed graph to give a initial partition. Such methods are known to be effective at finding high quality partitions and are outlined in Section 2.1. The levels are then iteratively unravelled by bursting apart the supernodes. Each resulting partition is refined using the Kernighan-Lin heuristic that is outlined in Section 2.2.

We first describe both techniques for partitioning a graph with $2n$ vertices into two equally sized subsets. Heuristics for solving this problem are the building blocks for heuristics that solve more general graph partitioning problems. For the graph $G = (V, E)$, suppose that V contains $2n$ vertices. We wish to partition V into two sets A and B, each containing n vertices, such that the number of edges joining vertices in A to vertices in B is minimized.

2.1 Spectral Partitioning Methods

Spectral partitioning heuristics generally give very high quality graph partitions. Let x be a n dimensional vector such that $x_i = \pm 1$ and $\sum_{i \in V} x_i = 0$. Consider the function

$$f(x) = \frac{1}{4} \sum_{(i,j) \in E} (x_i - x_j)^2.$$

If $x_i = 1$ when $x_i \in A$ and $x_i = -1$ when $x_i \in B$ then notice that $f(x)$ counts the number of edges crossing between the sets A and B. This is because $(x_i - x_j)^2$ is zero if x_i and x_j have the same sign and is equal to four if x_i and x_j differ in sign.

The adjacency matrix for the graph G is defined by

$$A_{i,j} = \begin{cases} 1 \text{ if } (i,j) \in E \\ 0 \text{ otherwise.} \end{cases}$$

The degree matrix for the graph G is a diagonal matrix $D = \text{diag}(d_i)$ where d_i is the number of edges incident on vertex i. We can write $f(x)$ in terms of the adjacency matrix and degree matrix for the graph G as follows. We note that

$$\sum_{(i,j) \in E} (x_i - x_j)^2 = \sum_{(i,j) \in E} (x_i^2 + x_j^2) - \sum_{(i,j) \in E} 2x_i x_j.$$

We can rewrite each of terms on the right hand side as follows

$$\sum_{(i,j)\in E}(x_i^2+x_j^2)=\sum_{(i,j)\in E}2=2|E|=\sum_{i\in V}x_i^2 d_i=x^TDx$$

and

$$\sum_{(i,j)\in E}2x_ix_j=\sum_{i\in V}\sum_{j\in V}x_iA_{ij}x_j=\sum_{i\in V}x_i\sum_{j\in V}A_{ij}x_j=x^TAx.$$

Thus, we may define a Laplacian matrix, $L = D - A$ and conclude that $f(x) = \frac{1}{4}x^TLx$. The graph partitioning problem can then be formulated as the following discrete minimization problem:

$$\begin{gathered}\min \tfrac{1}{4}x^TLx \\ \text{subject to } \textstyle\sum_{i=1}^n x_i = 0, \quad x_i = \pm 1.\end{gathered}$$

The crucial step in spectral methods is the relaxation of the discrete constraint $x_i = \pm 1$ in the following continuous minimization problem:

$$\begin{gathered}\min \tfrac{1}{4}x^TLx \\ \text{subject to } \textstyle\sum_{i=1}^n x_i = 0, \quad x^Tx = n.\end{gathered}$$

A solution x of this problem is projected onto the feasible region of the discrete problem to obtain an approximate solution of that problem.

Let $\Lambda_1, \Lambda_2, \ldots, \Lambda_n$ be an orthonormal basis of eigenvectors of L with the corresponding eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. We can therefore write $x = \sum_i \alpha_i\Lambda_i$ and so $\sum_i \alpha_i^2 = n$. It is possible to show that $\Lambda_1 = \frac{1}{\sqrt{n}}e$ and $\lambda_1 = 0$. By substituting for x we may write

$$f(x)=\frac{1}{4}(\sum_{i=2}^n\alpha_i^2\lambda_i).$$

because $\lambda_1 = 0$. Given the ordering of the eigenvalues, this gives

$$f(x)\geq\frac{1}{4}(\alpha_2^2+\alpha_3^2+\cdots+\alpha_n^2)\lambda_2\geq\frac{n\lambda_2}{4}.$$

Note also that $x^* = \sqrt{n}\Lambda_2$ achieves this lower bound. Furthermore, the balance constraint that $\sum_{i=1}^n x_i = 0$ is satisfied, since

$$e^Tx^*=(\sqrt{n}\Lambda_1)^T(\sqrt{n}\Lambda_2)=\Lambda_1^T\Lambda_2=0.$$

Since x^* satisfies the constraints of the continuous problem and minimizes $f(x)$, x^* is a solution to the continuous problem.

Spectral methods calculate the vector Λ_2 associated with the second smallest eigenvalue of the Laplacian matrix $L = D - A$. This eigenvector is called the Fiedler vector and is used to generate the following partition, based on the ideas outlined above. The vertices corresponding the the largest $\frac{n}{2}$ entries in the Fiedler vector are assigned to set A. The vertices corresponding to the smallest $\frac{n}{2}$ entries in the Fiedler vector are assigned to B. The Fiedler vector is typically computed using the Lanczos algorithm.

When $k = 2^i$, one may use spectral bisection to partition graphs into k subsets, by recursively bisecting the sets until reaching the desired number of sets. Spectral bisection has also been generalized to more general partitioning problems [16,13] using more information from the eigenvector decompostion.

2.2 Kernighan-Lin Heuristic

Kernighan and Lin give a effective heuristic for partitioning graphs so as to minimize the cost of the resulting partition [17]. Their heuristic is particularly effective when used to refine an already good partition but tends to break down when applied to a poorly partitioned graph.

A starting partition A, B of V is generated by a spectral partitioning method. Attempts are made to decrease the number of edges joining A to B by interchanging subsets of A and B. For each $a \in A$, define the *external cost* E_a as the number of edges in G joining a to vertices in B. Define an *internal cost* I_a as the number of edges joining a to other vertices in A. For each $b \in B$ define E_b and I_b similarly. For every $v \in V$ define D_v as the difference between external and internal costs, that is $D_v = E_v - I_v$. It can be shown that the gain from interchanging a vertex $a \in A$ with a vertex $b \in B$ is $D_a + D_b$ if there is no edge joining a and b and $D_a + D_b - 2$ if there is an edge joining a and b.

First, D_v is calculated for all $v \in V$. We define

$$\psi(a, b) = \begin{cases} 2 & \text{if } (a, b) \in E \\ 0 & \text{otherwise.} \end{cases}$$

Second, we choose $a \in A$, $b \in B$ such that

$$g_1 = D_a + D_b - \psi(a, b)$$

is maximized. We set this a and b aside for the time being and call them a_1 and

b_1 respectively. Next, the D_v are recalculated using the following formulae:

$$D_x \leftarrow D_x + \psi(x, a_1) - \psi(x, b_1), x \in A \setminus \{a_1\}$$
$$D_y \leftarrow D_y + \psi(y, b_1) - \psi(y, a_1), y \in B \setminus \{b_1\}.$$

Here, we are recalculating the differences D_v as if a_1 and b_1 have been removed from the graph. Next, we repeat the process by choosing $a_2 \in A \setminus \{a_1\}$ and $b_2 \in B \setminus \{b_1\}$ to maximize

$$g_2 = D_{a_2} + D_{b_2} - \psi(a_2, b_2).$$

The quantity g_2 is the additional gain that can be made by exchanging vertices a_2 and b_2 in addition to a_1 and b_1. We continue this procedure until all of the vertices in the sets A and B have been exhausted. Each time a pair of vertices a_k and b_k is identified, that pair is removed from consideration in future rounds. The size of the sets being considered decreases by one after each round, so that the procedure is performed a total of n rounds.

Finally, we choose k to maximize the sum $S = \sum_{i=1}^{k} g_i$. If $S > 0$ we can reduce the value of S by interchanging $a_1, a_2, \ldots, a_k$ with $b_1, b_2, \ldots, b_k$. Once, this is done, we can treat the resulting partition as the initial partition and start the heuristic again from the beginning. If $S = 0$ then the current partition is a locally optimum partition.

If at each round, the difference values D_x for $x \in A$ and D_y for $y \in B$ are kept in sorted order, then only a few contenders for pairs that maximize g_k need to be evaluated. When this is done, the heuristic runs in time proportional to $n^2 \log n$. Note that this is much more reasonable than enumerating all of the partitions of G.

Once the basic two-way partitioning heuristic is well understood, we can easily extend it to partitioning a set of $n = km$ vertices in k vertex sets in such a way that the number of edges between distinct vertex sets is minimized. We start with an arbitrary partition of the vertices into k equally sized subsets. The two-way partitioning heuristic is then applied to pairs of subsets until all subsets are pairwise optimal. There are $\binom{k}{2}$ pairs of subsets that must be considered. Note that more than one pass through the pairs of subsets may be necessary since, when two subsets are made optimal with respect to each other by means of interchanging vertices, this may change their optimality with respect to other subsets.

2.3 Unequally Sized Partitions

Suppose that we wish to partition a set of vertices into k subsets, but that we do not care whether or not each of the subsets has exactly the same number of vertices. We can then add enough *dummy vertices* to the problem, so that there will be a total of km' vertices in the problem. These dummy vertices have no edges incident on them. When the resulting problem is solved and the dummy vertices are removed from the subsets in the resulting partition, the resulting partition will consist of k subsets each containing between 0 and m' of the original n vertices.

Notice, that if one of the k subsets is empty, then we have essentially partitioned the n vertices into $k-1$ subsets. This indicates that we can also introduce slack into the number of subsets in the partitions. To generate a partition of between j and k subsets each containing possibly unequal numbers of vertices, simply introduce enough dummy vertices so that there is a total of $k\lceil n/j \rceil$ vertices in the resulting problem. We remove the dummy vertices from each of the subsets in the resulting locally optimal solution and then discard any subsets in the partition that are empty.

2.4 Matrix Partitioning

We now discuss how the graph partitioning heuristics outlined above can be used to partition a matrix into arrowhead form. First, the graph of the matrix is formed and enough dummy vertices are added to reflect the amount of slack we desire in both the number of blocks and the uniformity of size for the blocks. The spectral method is applied to the resulting graph. Then the resulting graph is locally refined using the Kernighan-Lin heuristic.

We are then left with a partition of vertices. We examine the edges that join vertices in distinct subsets of the partition. For each vertex v we count the number of edges connecting the vertex to vertices outside of the subset in the partition containing v. Call this number E_v, the external cost of the vertex v. We apply a greedy algorithm that looks for the largest E_v and removes that vertex from the graph. The E_v are then recalculated for the resulting graph. Actually, this recalculation is easy, since we only need decrement E_w for all vertices w coincident on an edge with the vertex v. We continue this procedure until all E_v in the remaining graph are zero. In a tie breaking procedure we favor removing rows to columns.

It is possible to improve the partitions that this heuristic generates, but the added cost appears significant. Each time a vertex is removed the Kernighan-Lin heuristic can be re-applied to the remaining graph. Although some improvement was noted for many of the NETLIB problems, this procedure typically increased the running time by a factor of 10. For applications that are frequently repeated, the partition improvement may warrant this extra computation. However, in our applications to parallel solution of linear programs, it was not worthwhile.

The column vertices removed during this procedure correspond to columns in the right-hand border in our matrix partition. The row vertices removed during this procedure correspond to rows in the lower border in our matrix partition. The subsets in the original graph partition are now completely disconnected from each other, for all edges connecting one subset to another have been removed. Each of these subsets forms a block in the matrix partition. This completes the transformation to arrowhead form.

It is relatively easy to transform a matrix in arrowhead form into a singly-bordered block-diagonal form. To accomplish this, we consider the variables corresponding to the linking columns

$$\begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ C_K \\ D \end{pmatrix}.$$

For each column j of this matrix, we introduce multiple copies of the corresponding variable, one copy for each block C_i (or D) that has at least one nonzero in column j. These multiple copies are used to decouple the corresponding C_i's. We then add coupling constraints that force these variables all to be equal. This technique is the same as one used in stochastic programming to treat non-anticipativity (see [23]). Other techniques are described in [20]. For example,

$$\begin{pmatrix} x & x & & & x \\ x & x & & & x \\ & & x & x & x \\ & & x & x & \\ x & & x & & \end{pmatrix}$$

gets transformed into

$$\begin{pmatrix} x & x & & & x & \\ x & x & & & x & \\ & & x & x & & x \\ & & x & x & & \\ x & & x & & & \\ & & & & 1 & -1 \end{pmatrix}$$

or with a single column permutation that makes the 5th column the 3rd column

$$\begin{pmatrix} x & x & x & & & \\ x & x & x & & & \\ & & & x & x & x \\ & & & x & x & \\ x & & & x & & \\ & & 1 & & & -1 \end{pmatrix}.$$

Note that at most $p \times K$ constraints are added if C is completely dense, many fewer if C is sparse.

3 Partitioning Results

In this section, we present some computational results to demonstrate the effectiveness of the heuristics outlined above. We coded the heuristics ourselves and also tested two publicly available graph partitioning routines, Chaco[12] and Metis[15]. Almost without exception, Metis performed the best in terms of run time and partition quality. For this reason we use Metis for all the results listed in this paper. We give three sets of results to show how well our partitioning heuristics perform *as partitioning algorithms.* That is, how close do they come to producing a matrix in arrowhead form with the desired number of blocks. We have run the above matrix partitioning procedure on all the sample linear programs that are publicly available via anonymous ftp from netlib.att.com (see [7]). We have attempted to partition each problem into 2, 4, 8, 16, 32, 64, 128, and 256 blocks.

Let m_i denote the number of rows and n_i the number of columns in the i^{th} block. Let M and N denote the number of rows and columns in the original matrix. Throughout this section we use the following measure to determine the effectiveness of our partition into K blocks:

$$\mu_{p,q} := p\alpha + q\beta$$

where $p + q = 1$ and

$$\begin{aligned}
m^* &:= \max_{1\le i\le K} m_i, \\
n^* &:= \max_{1\le j\le K} n_j, \\
\alpha &:= \frac{1}{K^2}\sum_{i=1}^{K}\frac{m_i}{m^*}\sum_{j=1}^{K}\frac{n_j}{n^*}, \\
\beta &:= \frac{\sum_{i=1}^{K} m_i \sum_{j=1}^{K} n_j}{MN}.
\end{aligned}$$

We note that α is equal to one if each of the blocks has an equal number of rows and an equal number of columns and diminishes to zero as the numbers of rows and columns become increasingly variable. The value of β simply measures the fraction of the partition that is not part of the lower or right hand border. That means that $1 - \beta$ measures the fraction of the partition that is made up of blocks. Thus, if we manage to split the matrix into K blocks of equal area, then $\mu_{p,q} = 1$. If the blocks are of unequal area, then μ decreases. We may control the extent to which coupling constraints and linking variables are penalized by adjusting the parameters p and q. Values of q near one (p near zero) will penalize linking constraints heavily, while values of q near zero (p near one) will penalize unevenly sized blocks. The values $p = 0.1$, $q = 0.9$ were chosen to try to reflect how the partitioning would enable parallel solution of the underlying linear program. Unequal sized blocks probably lead to load balancing problems, while linking constraints are usually treated by some synchronization procedure, leading to loss of parallel efficiency. In both of these cases, the resulting $\mu_{p,q}$ becomes closer to 0. Our experience indicates that loss of parallel efficiency is a much more critical problem than load balancing, so we penalized the number of linking constraints rather severely.

In the first set of computational results, we show the effect of changing the number of dummy nodes in the problem and use this analysis to fix this parameter for the remainder of our computation. We fix the number of requested blocks at 8 and vary the number of dummy nodes to be 0, 20, and 40 percent of the number of nodes in the original problem. The results are given in Table 2, Table 3 and Table 4. On a large subset of the problems, the resulting values of μ are greater than 0.6. Figure 2 through Figure 5 show the original matrix, the resulting permuted matrices and corresponding values of μ for a particular problem. We believe this shows that our heuristic performs very well.

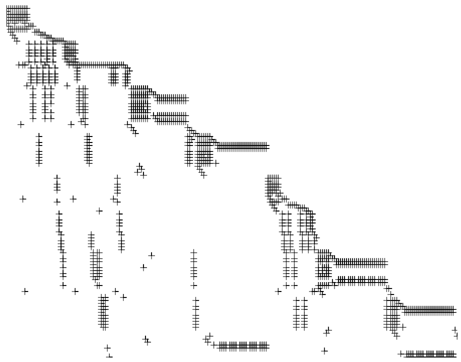

Fig. 2. Share1b – Original Structure

In most of the problems that are amenable to partitioning, adding 20% dummy nodes improves the partition. However, adding 40% dummy nodes sometimes degrades the resulting partition. The reason for this is that adding too many dummy nodes tends to make eigenvalues clump together in spectral methods and causes the quality of the resulting partitions to degrade. We stress that using spectral methods followed by refinement via Kernighan-Lin is important when dummy nodes are present. We initially tried a Kernighan-Lin heuristic alone that generally resulted in partitions that were poorer at the 20% level and even worse at the 40% level. Too many dummy nodes encourages Kernighan-Lin methods to fall into non-global optima much more frequently.

There are cases however, where increasing the number of dummy nodes does seem to be beneficial. A good example of this is the problem named 'share1b'. We present some graphical representations of this problem in Figure 2 through Figure 5 as the number of dummy variables is increased. Notice how there is a tradeoff between making the number of linking constraints and variables small and keeping a fairly regular block size. In many cases, one can see exactly where linking variables were inserted into blocks as the number of dummy nodes is increased.

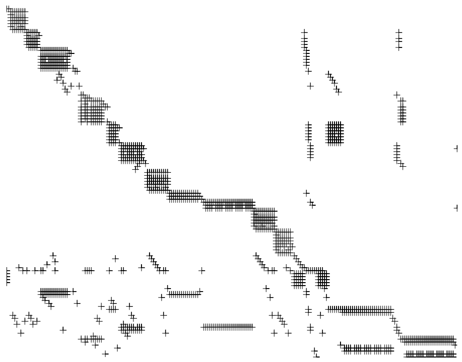

Fig. 3. Share1b – 8 Block Partition, 0 % Dummy Variables ($\mu = 0.63$)

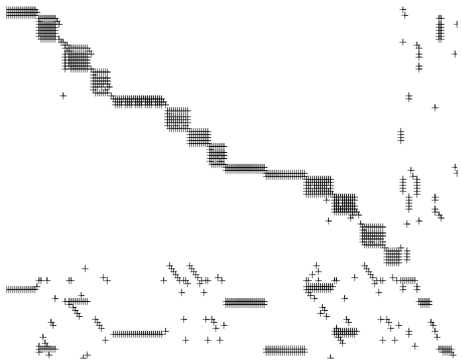

Fig. 4. Share1b – 8 Block Partition, 20 % Dummy Variables ($\mu = 0.65$)

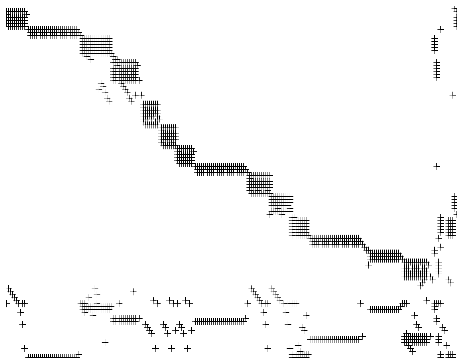

Fig. 5. Share1b – 8 Block Partition, 40 % Dummy Variables ($\mu = 0.72$)

In the second set of results, we show how well our heuristic identifies hidden structure in a problem. To do this, we consider the Patient Distribution System problem [1] which has a natural 11 block structure (see Figure 6) and was obtained from the United States Air Force. The problem we consider here is of size 1,386 by 3,729, with 11 blocks all approximately 125 by 340 with 90 coupling constraints. This structure is hidden by randomly permuting its rows and columns (see Figure 7). Our algorithm is then applied to this matrix and the resulting matrix is shown in Figure 8. Note that although 16 blocks were requested, our algorithm returned the natural 11 block structure since we gave the graph partitioning algorithm approximately 2,500 dummy nodes. The easier problem of finding 11 blocks also finds a similar form without any difficulties. These results show that our algorithm effectively detects arrowhead form when it exists. For this problem, our heuristic takes 1.7 seconds of CPU time on a Sparc 2 for the 16 block request. As we shall see in Section 4.2, this is a very small fraction of the time needed for solving the underlying linear program on a parallel machine. This solution time is typical for most of the problems that we have encountered. There are a few problems which take longer (the worst is 25 seconds).

For the remainder of the results in this section, we fix the level of dummy variables at 20 percent. The μ values for different numbers of blocks for various of the problems from the NETLIB collection are given in Table 5, Table 6 and Table 7. In most cases, the μ value decreases as the number of blocks increase as would be expected. There are exceptions to this rule since the heuristic does not always give a globally optimal solution to the partitioning problem. Some of the problems in the NETLIB suite do not split effectively into more than 8 or 16 blocks due to their relative density. Further, our algorithm is much more effective on very large and sparse problems, as would be expected.

Figure 9 through Figure 17 plot the partitioned matrix for the problem 'stocfor2' using 20% dummy variables. Note how effective the method appears to be in generating the blocks and the increase in linking variables and coupling constraints for more blocks.

Certainly different solution techniques for linear and nonlinear programs will require measures other than μ to find what is the best partitioning. The greedy technique for partitioning can easily be modified to generate other partitionings if this is necessary (for example, if linking variables are less costly than coupling constraints the tie break could favor variables over constraints, etc). Finally, we note that a-priori removal of dense rows and columns from the matrix did not improve the resulting partition when using our heuristics.

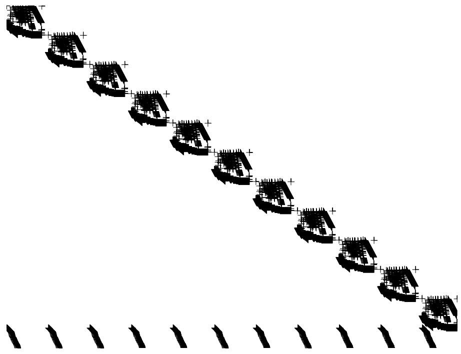

Fig. 6. PDS Problem – Original Structure

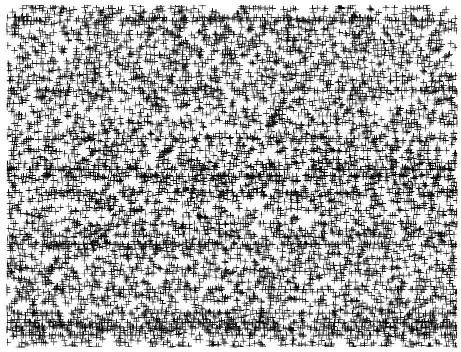

Fig. 7. PDS Problem – Randomly Permuted

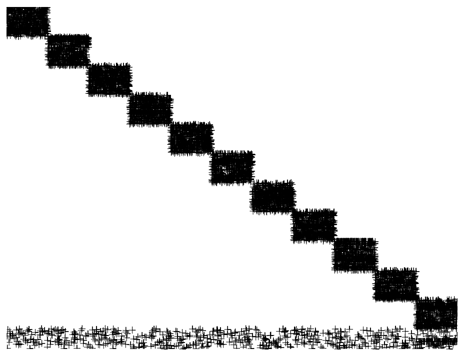

Fig. 8. PDS Problem – Result of Partitioning ($\mu = 0.94$)

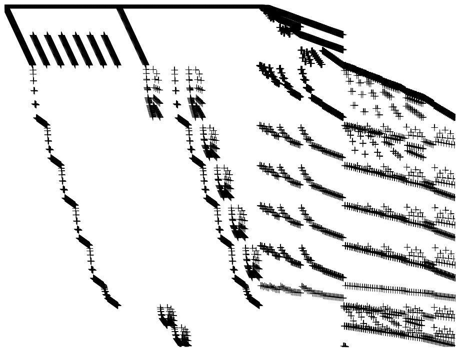

Fig. 9. Stocfor2 – Original Structure

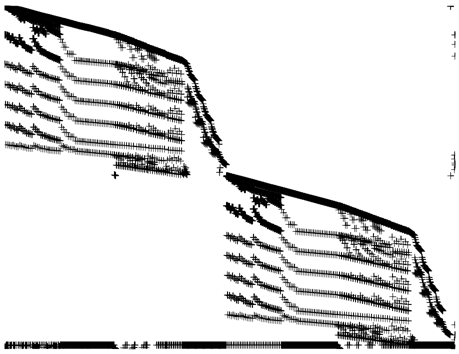

Fig. 10. Stocfor2 – 2 Block Partition($\mu = 0.90$)

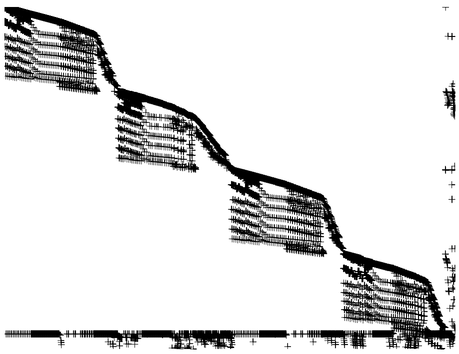

Fig. 11. Stocfor2 – 4 Block Partition ($\mu = 0.80$)

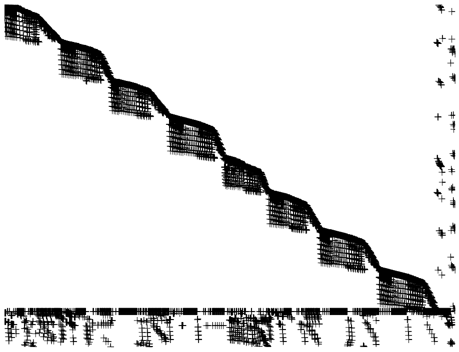

Fig. 12. Stocfor2 – 8 Block Partition ($\mu = 0.73$)

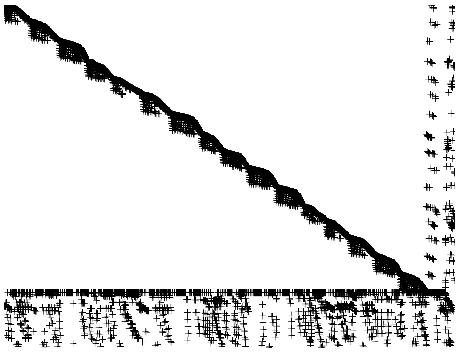

Fig. 13. Stocfor2 – 16 Block Partition ($\mu = 0.73$)

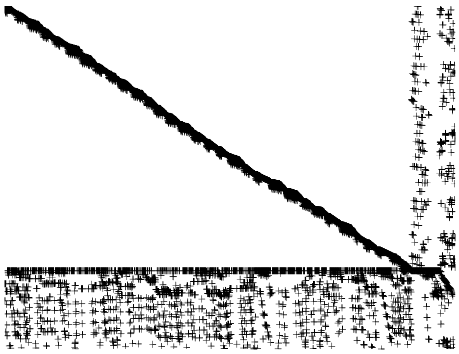

Fig. 14. Stocfor2 – 32 Block Partition ($\mu = 0.69$)

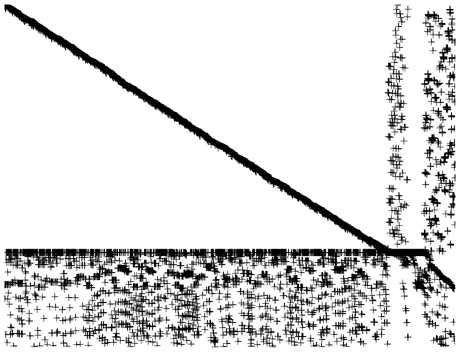

Fig. 15. Stocfor2 – 64 Block Partition ($\mu = 0.63$)

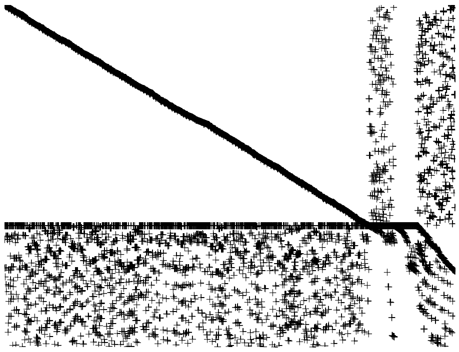

Fig. 16. Stocfor2 – 128 Block Partition ($\mu = 0.55$)

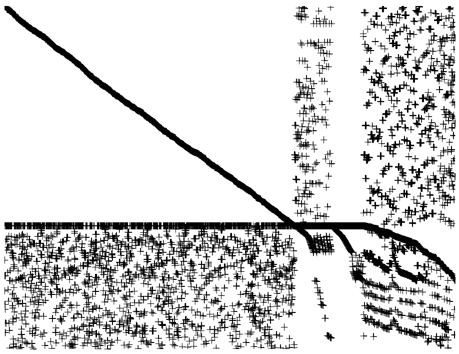

Fig. 17. Stocfor2 – 256 Block Partition ($\mu = 0.44$)

4 Parallel Solution of Linear Programs

The remainder of this paper is concerned with the utility of the aforementioned partitioning algorithm. We apply the matrix partitioning scheme to linear programming problems arising in the NETLIB collection [7] to form a singly-bordered block-diagonal linear program (see Section 2). We then apply a variant of the bundle method to an appropriately formed dual problem and implement the resulting algorithm on the Thinking Machines CM-5 to obtain an efficient parallel method for general linear programming problems.

After partitioning, the linear programming problem that we solve has the form

$$\begin{aligned} \min_{x=(x_1,\dots,x_K)} \ & \sum_{i=1}^{K} c_i^T x_i \\ \text{subject to } \ & B_i x_i = b_i, \quad x_i \in X_i \\ & \sum_{i=1}^{K} R_i x_i = r. \end{aligned}$$

Note that simple bound constraints on the variables have been represented as $x_i \in X_i$. There are several known techniques for solving problems of this form in parallel [3,30,31,11,14]. We now outline the method that we use in this paper.

4.1 Bundle-Level Decomposition

For notational simplicity, we let

$$f_i(x_i) := \begin{cases} c_i^T x_i & \text{if } B_i x_i = b_i,\ x_i \in X_i \\ +\infty & \text{otherwise,} \end{cases}$$

and we note that f_i is a closed convex function, proper if the ith block is feasible. Our problem is rewritten as

$$\min_{x=(x_1,\dots,x_K)} \left\{ \sum_{i=1}^{K} f_i(x_i) : \sum_{i=1}^{K} R_i x_i = r. \right\}$$

Following Robinson [27,26], we introduce a perturbation function

$$F(x,p) := \begin{cases} \sum_{i=1}^{K} f_i(x_i) & \text{if } r - \sum_{i=1}^{K} R_i x_i = p, \\ +\infty & \text{otherwise,} \end{cases}$$

a Lagrangian

$$L(x,y) := \inf_p \{y^T p + F(x,p)\} = y^T r + \sum_{i=1}^{K} \{f_i(x_i) - y^T R_i x_i\},$$

and a dual problem

$$\sup_y g(y), \tag{1}$$

where

$$g(y) := \inf_x L(x,y) = y^T r - \sum_{i=1}^{K} f_i^*(R_i^T y),$$

with

$$f_i^*(R_i^T y) = \sup_{x_i} \{y^T R_i x_i - f_i(x_i)\}.$$

Note that, for a given value of y, $f_i^*(R_i^T y)$ is easily calculated by solving the following linear program, the dimension of which is the size of the corresponding block B_i

$$\max_{x_i} \left\{ (y^T R_i - c_i^T) x_i : B_i x_i = b_i, x_i \in X_i. \right\} \tag{2}$$

Under the constraint qualification

$$r \in \sum_{i=1}^{K} R_i (\text{ridom} f_i^*),$$

the dual problem (1) has a solution and the dual optimal value is equal to the primal optimal value. Thus we solve the dual problem (1), whose dimension is given by the number of coupling constraints q.

Note that g is a concave function, but it is not necessarily differentiable. However, it is possible (under the condition $\bigcap_{i=1}^{K} (\text{im} R_i^T \cap \text{ridom} f_i^*) \neq \emptyset$) to determine at least one subgradient of $(-)g$ using

$$\partial g(y) = r - \sum_{i=1}^{K} R_i \partial f_i^*(R_i^T y),$$

where

$$\partial f_i^*(R_i^T y) = \arg\min\{f_i(x_i) - y^T R_i x_i\},$$

as shown in [28, p. 223]. Thus a subgradient at y of $(-)g$ can be calculated by solving the K subproblems (2). Therefore, to solve (1) we use the bundle-level algorithm from [18], which is now discussed in more detail. Other related work on bundle methods can be found in [19,30,21,2].

Suppose that we wish to

$$\min_{x \in Q} f(x),$$

where f is a convex function and Q represents some simple convex constraint set. The algorithm builds a piecewise linear convex "model" function m which underestimates f and is given by

$$m(x) := \max_{j=1,\dots,i}\{f(x^j) + f'(x^j)(x - x^j)\},$$

where $f'(x^j) \in \partial f(x^j)$ and x^j are the points the algorithm has already visited. Note that superscripts on the x represent different vectors in $\mathbb{R}^N$, whereas subscripts refer to component vectors of x. We can therefore calculate a lower and upper bound on the optimal value of f by evaluating

$$\begin{aligned} f_* &= \text{ minimum value of model } m \text{ over } Q, \\ f^* &= \text{ minimum function value already seen.} \end{aligned}$$

Associated with f^* is an attaining x^*. The algorithm chooses the next point at which to evaluate the function and a subgradient by projecting x^* onto a carefully chosen level set of the model function m. The "level" L is adjusted depending on how well the algorithm is progressing. A full description is now given.

Given $x^1 \in Q$ and $\lambda \in (0,1)$, let $\Delta'_0 = \infty$. Having x^i, repeat the following steps until convergence is attained:

(i) Calculate $f(x^i)$ and $f'(x^i) \in \partial f(x^i)$.

(ii) Evaluate f_*, f^* and x^* and let $\Delta = f^* - f_*$.
(iii) Let $L' = \lambda f_* + (1-\lambda) f^*$ and determine the new level by

$$L = \begin{cases} L' & \text{if } \Delta < \lambda \Delta'_{i-1}, \\ \min\{L', L\} & \text{otherwise,} \end{cases}$$

where

$$\Delta'_i = \begin{cases} \Delta & \text{if } \Delta < \lambda \Delta'_{i-1}, \\ \Delta'_{i-1} & \text{otherwise.} \end{cases}$$

(iv) Project x^* onto the level set of the model $M_L = \{x \in Q | m(x) \leq L\}$, that is

$$x^{i+1} = \pi(x^* | M_L).$$

It can be shown (see [18]) that this technique will generate function values arbitrarily close to the optimal value under a simple compactness assumption on Q. Each iteration requires the evaluation of $f(x^i)$ and $f'(x^i)$ which can be carried out in parallel in our work as described above. The synchronization requires the solution of a simple linear program and projection problem, both over the same feasible set. This can be carried out very easily using crash techniques and restarts. The key to the success of this approach is a partition with roughly equal sized blocks and few coupling constraints.

4.2 Parallel Implementation

The algorithm for solving linear programs given in the previous section has been implemented in PVM C libraries [8] on the Thinking Machines CM-5 supercomputer and used to solve a variety of the largest linear programs in the NETLIB collection [7].

PVM [8] is a set of C libraries that facilitate parallel programming in a message passing environment. The libraries are portable to a variety of parallel computers and also function well in distributed parallel applications.

In our implementation, message passing is used to handle data associated with the linear and quadratic programs used for synchronization. The first part of our code partitions the constraints of the problems according to the algorithm given in Section 2. The output of this phase are two permutations, one for the constraints and one for the variables, the application of which gives the constraint matrix an arrowhead form. In determining these permutations, we treat all the constraints as if they were equalities and do not add slack variables since it is extremely likely that the slack variables would be added

Table 1
Parallel Solution Statistics

Problem	% Density	Dual Dim	32 Block μ	Iters	% Efficiency
sc205	1.30	92	0.62	7	89.7
scfxm3	0.60	348	0.74	10	91.0
sierra	0.40	409	0.75	13	90.7
scagr25	0.90	321	0.75	10	80.3
bnl2	0.20	148	0.66	9	88.1
sctap2	0.40	273	0.80	12	85.2
sctap3	0.30	520	0.74	11	83.5
stocfor2	0.22	240	0.69	11	87.2
scorpion	1.20	76	0.92	7	90.6
gfrd-pnc	0.50	164	0.79	10	87.3

to the constraint blocks that we generate anyway, and the extra preprocessing work is not justified. This hypothesis could be tested in future work.

Once the partitioning is complete, we apply the bundle-level method to the resulting linear program. For the function and gradient evaluation steps we use an implementation of the revised simplex method written in C which incorporates the Reid basis updating technique [25] and other computational enhancements [24]. The synchronization steps solve the linear programs using the same code as the parallel steps, the quadratic program resulting from the projection is solved using a method due to Mifflin [22].

Special care is taken during the synchronization step to ensure that the vector y is indeed a subgradient. If a y is generated such that $R_i^T y \notin \mathrm{dom} f_i^*$, then the level is reduced by a multiplicative factor and the resulting problems are resolved in parallel until a suitable y is found. Using this technique we have circumvented the compactness assumptions required by the theory. This is not guaranteed to work, but has proven very effective in our computations.

In Table 1 we report the results on the subset of the NETLIB problems that had very good μ values. We give problem density, the 32 block μ value calculated by our algorithm, the number of steps that the bundle-level method took to solve the problem on 32 processors and the parallel speedup efficiency. Our termination criterion required that two successive iterations have objective function values within 10^{-9} of each other.

The speedups for all of the problems are rather good. It should be noted that as the number of linking constraints grows, the efficiency decreases due to the difficulty of treating such constraints. Contrary to popular belief, however, the bundle-level method would appear to be a promising approach for solving such structured problems. Further computational comparison is needed between the bundle-level method and the other methods mentioned elsewhere in this paper, but this is beyond the scope of this work.

5 Conclusions

We have shown how to reorder the variables and constraints of a mathematical program in order to detect underlying arrowhead structure. The technique uses graph partitioning algorithms on the associated graph of the constraint matrix. We demonstrated the effectiveness of our heuristics on the NETLIB problems and solved several of the larger problems with high parallel efficiency.

We recommend the following 3 techniques.

(i) Multilevel partitioning algorithms that combine spectral methods and the Kernighan-Lin heuristic, as implemented in [15,12].
(ii) Adding 20 % dummy nodes to the graph to allow uneven partitions.
(iii) Bundle-level methods for solving the resulting mathematical programs.

References

[1] W. J. Carolan, J. E. Hill, J. L. Kennington and S. N. an S. J. Wichmann, "An empirical evaluation of the KORBX algorithms for military airlift applications", *Operations Research* 38 (1990) 240–248.

[2] B. J. Chun, S. J. Lee and S. M. Robinson, "An implementation of the bundle decomposition algorithm", Tech. Rep. 91-6, Department of Industrial Engineering, University of Wisconsin (Madison, Wisconsin, 1991).

[3] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations Research* 8 (1960) 101–111.

[4] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices* (Oxford University Press, Oxford, England, 1986).

[5] M. C. Ferris and O. L. Mangasarian, "Parallel constraint distribution", *SIAM Journal on Optimization* 1 (1991) 487–500.

[6] M. C. Ferris and O. L. Mangasarian, "Parallel variable distribution", *SIAM Journal on Optimization* 4 (1994) 815–832.

[7] D. M. Gay, "Electronic mail distribution of linear programming test problems", *COAL Newsletter* 13 (1985) 10–12.

[8] G. A. Geist, A. L. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek and V. S. Sunderam, *PVM: Parallel Virtual Machine* (MIT Press, Cambridge, Massachusetts, 1994).

[9] A. George, "An automatic one-way dissection algorithm for irregular finite-element problems", *SIAM J. Numer. Anal.* (1980) 740–751.

[10] J. R. Gilbert and E. Ng, "Predicting structure in nonsymmetric sparse matrix factorizations", in: A. George, J. Gilbert and J. Liu eds., *Graph Theory and Sparse Matrix Computation* (Springer-Verlag, 1993) .

[11] R. V. Helgason, J. L. Kennington and H. A. Zaki, "A parallelization of the simplex method", *Annals of Operations Research* 14 (1988) 17–40.

[12] B. Hendrickson and R. Leland, "The Chaco user's guide: Version 2", Technical Report SAND94-2692, Sandia National Laboratories (Albuquerque, New Mexico, 1994).

[13] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs", in: *Proceedings of Supercomputing '95* (1995).

[14] J. K. Ho, T. C. Lee and R. P. Sundarraj, "Decomposition of linear programs using parallel computation", *Mathematical Programming* 42 (1988) 391–405.

[15] G. Karypis and V. Kumar, "METIS: Unstructured graph partitioning and sparse matrix ordering system, version 2.0", Technical report, University of Minnesota, Department of Computer Science (Minnesota, 1995).

[16] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs", Technical Report 95-064, University of Minnesota, Department of Computer Science (Minnesota, 1995).

[17] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs", *The Bell System Technical Journal* (1970) 291–307.

[18] C. Lemaréchal, A. Nemirovskii and Y. Nesterov, "New variants of bundle methods", *Mathematical Programming* 69 (1995) 111–147.

[19] C. Lemaréchal, J. J. Strodiot and A. Bihain, "On a bundle algorithm for nonsmooth optimization", in: O. L. Mangasarian, R. R. Meyer and S. M. Robinson eds., *Nonlinear Programming*, vol. 4 (Academic Press, 1981) pp. 245–282.

[20] D. Medhi, *Decomposition of Structured Large-Scale Optimization Problems and Parallel Optimization* (Ph.D. thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1987), Technical Report 718.

[21] D. Medhi, "Parallel bundle-based decomposition for large-scale structured mathematical programming problems", *Annals of Operations Research* 22 (1990) 101–127.

[22] R. Mifflin, "A stable method for solving certain constrained least squares problems", *Mathematical Programming* (1979) 141–158.

[23] J. M. Mulvey and A. Ruszczynski, "A diagonal quadratic approximation methods for large scale linear programs", *Operations Research Letters* 12 (1992) 205–215.

[24] J. L. Nazareth, *Computer Solution of Linear Programs* (Oxford University Press, Oxford, 1987).

[25] J. K. Reid, "A sparsity-exploiting variant of the Bartels-Golub decompostion for linear programming bases", *Mathematical Programming* 24 (1982) 55–69.

[26] S. M. Robinson, "Bundle-based decomposition: Description and preliminary results", in: A. Prékopa, J. Szelezcán and B. Straazicky eds., *System Modeling and Optimization*, vol. 84 of *Lecture Notes in Control and Information Sciences* (Springer-Verlag, Berlin, 1986) pp. 751–756.

[27] S. M. Robinson, "Bundle-based decomposition: Conditions for convergence", in: H. Attouch, J. P. Aubin, F. Clarke and I. Ekeland eds., *Analyse Non Linéaire* (Gauthier-Villaars, Paris, 1989) pp. 435–447.

[28] R. T. Rockafellar, *Convex Analysis* (Princeton University Press, Princeton, New Jersey, 1970).

[29] A. Ruszczynski, "On the regularized decomposition for stochastic programming problems", in: K. Marti and P. Kall eds., *Stochastic Programming: Numerical Techniques and Engineering Applications*, vol. 425 of *Lecture Notes in Control and Information Sciences* (Springer-Verlag, Berlin, 1995) pp. 93–108.

[30] H. Schramm and J. Zowe, "A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results", *SIAM Journal on Optimization* 2 (1992) 121–152.

[31] G. L. Schultz and R. R. Meyer, "An interior point method for block angular optimization", *SIAM Journal on Optimization* 1 (1991) 583–602.

Table 2
8 block partitions with varying percentages of dummy nodes

Problem	Dnsty	0% dummy nodes			20% dummy nodes			40% dummy nodes		
	%	α	β	μ	α	β	μ	α	β	μ
25fv47	0.9	0.61	0.64	0.64	0.65	0.65	0.65	0.74	0.74	0.74
80bau3b	0.1	0.46	0.57	0.56	0.59	0.61	0.60	0.65	0.62	0.62
adlittle	8.4	0.52	0.47	0.47	0.60	0.49	0.50	0.65	0.54	0.55
afiro	9.8	0.41	0.47	0.46	0.54	0.56	0.56	0.62	0.64	0.64
agg	3.2	0.44	0.65	0.63	0.46	0.67	0.65	0.46	0.66	0.64
agg2	2.9	0.42	0.69	0.66	0.49	0.74	0.71	0.46	0.74	0.71
agg3	2.9	0.42	0.70	0.67	0.53	0.87	0.83	0.46	0.75	0.72
bandm	1.8	0.81	0.65	0.67	0.94	0.69	0.71	0.98	0.72	0.74
beaconfd	7.6	0.31	0.42	0.41	0.37	0.50	0.48	0.38	0.50	0.49
blend	8.4	0.55	0.56	0.56	0.68	0.61	0.62	0.61	0.61	0.61
bnl1	0.8	0.64	0.69	0.69	0.65	0.69	0.69	0.71	0.71	0.71
bnl2	0.2	0.64	0.78	0.76	0.65	0.79	0.78	0.71	0.80	0.79
boeing1	2.9	0.48	0.57	0.56	0.66	0.69	0.69	0.67	0.70	0.70
boeing2	6.6	0.31	0.37	0.37	0.37	0.39	0.39	0.41	0.41	0.41
bore3d	2.1	0.56	0.62	0.62	0.63	0.65	0.65	0.59	0.62	0.62
brandy	4.7	0.45	0.50	0.50	0.54	0.57	0.57	0.56	0.58	0.58
capri	1.9	0.49	0.58	0.57	0.53	0.60	0.59	0.51	0.59	0.58
cycle	0.4	0.66	0.75	0.74	0.90	0.76	0.77	1.00	0.81	0.83
czprob	0.4	0.24	0.49	0.46	0.30	0.60	0.57	0.25	0.50	0.48
d2q06c	0.3	0.68	0.72	0.72	0.85	0.80	0.80	0.75	0.78	0.78
d6cube	1.8	0.21	0.02	0.04	0.23	0.02	0.04	0.28	0.02	0.05
degen2	1.9	0.55	0.48	0.49	0.58	0.49	0.50	0.70	0.51	0.53
degen3	1.0	0.56	0.49	0.49	0.72	0.52	0.54	0.87	0.55	0.58
dfl001	0.1	0.81	0.58	0.60	1.00	0.73	0.76	1.00	0.87	0.88
e226	4.4	0.47	0.63	0.62	0.55	0.64	0.63	0.61	0.66	0.66
etamacro	0.9	0.71	0.53	0.55	0.86	0.60	0.62	0.74	0.54	0.56
fffff80	1.40	0.34	0.46	0.45	0.35	0.47	0.45	0.35	0.46	0.45
finnis	0.9	0.74	0.66	0.67	0.88	0.79	0.80	0.79	0.70	0.71
fit1d	56.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fit1p	1.0	0.17	0.11	0.12	0.19	0.11	0.12	0.23	0.13	0.14
fit2d	50.6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3
8 block partitions with varying percentages of dummy nodes (continued)

Problem	Dnsty	0% dummy nodes			20% dummy nodes			40% dummy nodes		
	%	α	β	μ	α	β	μ	α	β	μ
fit2p	0.1	0.10	0.01	0.02	0.12	0.01	0.02	0.11	0.01	0.02
forplan	27.7	0.30	0.83	0.78	0.33	0.89	0.84	0.36	0.93	0.87
ganges	0.3	0.75	0.84	0.84	0.84	0.89	0.88	0.96	0.95	0.95
gfrd-pnc	0.5	0.72	0.83	0.82	0.81	0.84	0.84	0.75	0.83	0.82
greenbea	0.2	0.74	0.72	0.72	0.77	0.73	0.73	0.81	0.74	0.75
greenbeb	0.2	0.74	0.72	0.72	0.85	0.78	0.79	0.94	0.79	0.81
grow15	2.9	0.77	0.65	0.66	0.82	0.66	0.68	0.94	0.75	0.77
grow22	2.0	0.54	0.56	0.56	0.69	0.6	0.61	0.77	0.61	0.62
grow7	6.2	0.50	0.46	0.47	0.58	0.48	0.49	0.63	0.53	0.54
israel	9.5	0.49	0.43	0.44	0.64	0.52	0.53	0.74	0.59	0.60
kb2	16.1	0.27	0.34	0.33	0.28	0.35	0.34	0.28	0.35	0.34
lotfi	2.3	0.59	0.56	0.56	0.69	0.60	0.61	0.64	0.60	0.60
maros	0.8	0.62	0.71	0.70	0.79	0.88	0.87	0.68	0.76	0.76
nesm	0.7	0.67	0.59	0.60	0.77	0.62	0.64	0.81	0.65	0.67
perold	0.7	0.74	0.60	0.61	0.88	0.71	0.73	0.90	0.71	0.73
pilot	0.8	0.42	0.47	0.47	0.52	0.51	0.51	0.47	0.51	0.51
pilot.ja	0.8	0.68	0.61	0.61	0.69	0.61	0.62	0.76	0.63	0.64
pilot.we	0.5	0.60	0.72	0.71	0.61	0.73	0.72	0.66	0.74	0.73
pilot4	1.3	0.74	0.61	0.62	1.00	0.74	0.77	1.00	0.75	0.78
pilot87	0.7	0.49	0.48	0.48	0.59	0.50	0.51	0.65	0.53	0.54
pilotnov	0.6	0.61	0.62	0.62	0.69	0.65	0.65	0.64	0.62	0.62
recipe	4.5	0.71	0.79	0.78	0.86	0.91	0.90	0.88	0.91	0.91
sc105	2.6	0.65	0.75	0.74	0.70	0.77	0.76	0.67	0.76	0.75
sc205	1.3	0.85	0.82	0.82	1.00	0.83	0.85	1.00	0.89	0.90
sc50a	5.5	0.58	0.65	0.64	0.71	0.79	0.78	0.60	0.67	0.66
sc50b	5.1	0.46	0.61	0.60	0.57	0.67	0.66	0.51	0.66	0.64
scagr25	0.9	0.79	0.84	0.84	0.88	0.85	0.86	1.00	1.00	1.00
scagr7	3.0	0.66	0.70	0.70	0.70	0.71	0.71	0.84	0.75	0.75
scfxm1	1.7	0.63	0.70	0.69	0.81	0.74	0.75	0.98	0.79	0.81
scfxm2	0.9	0.72	0.82	0.81	0.94	1.00	0.99	1.00	1.00	1.00

Table 4
8 block partitions with varying percentages of dummy nodes (continued)

Problem	Dnsty	0% dummy nodes			20% dummy nodes			40% dummy nodes		
	%	α	β	μ	α	β	μ	α	β	μ
scfxm3	0.6	0.83	0.86	0.86	0.97	0.88	0.89	1.00	0.90	0.91
scorpion	1.2	0.85	0.86	0.86	1.00	0.97	0.97	0.88	0.87	0.87
scrs8	0.7	0.51	0.80	0.77	0.52	0.81	0.78	0.52	0.80	0.78
scsd1	5.3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
scsd6	2.8	0.17	0.16	0.16	0.19	0.17	0.17	0.18	0.16	0.16
scsd8	1.0	0.78	0.71	0.72	0.89	0.72	0.74	1.00	0.83	0.85
sctap1	1.4	0.84	0.71	0.72	1.00	0.73	0.76	1.00	0.83	0.85
sctap2	0.4	0.88	0.80	0.81	1.00	0.89	0.91	0.98	0.82	0.83
sctap3	0.3	0.89	0.82	0.83	0.99	0.88	0.89	1.00	0.92	0.93
seba	0.9	0.17	0.20	0.20	0.19	0.21	0.21	0.22	0.23	0.23
share1b	4.5	0.46	0.65	0.63	0.52	0.66	0.65	0.74	0.72	0.72
share2b	9.5	0.71	0.71	0.71	0.74	0.72	0.72	0.78	0.73	0.74
shell	0.5	0.54	0.59	0.59	0.58	0.60	0.60	0.66	0.68	0.68
ship04l	1.1	0.31	0.16	0.17	0.40	0.17	0.19	0.44	0.17	0.20
ship04s	1.1	0.32	0.41	0.40	0.37	0.43	0.42	0.40	0.47	0.46
ship08l	0.6	0.98	0.97	0.97	1.00	1.00	1.00	1.00	1.00	1.00
ship08s	0.6	0.53	0.82	0.79	0.56	0.84	0.82	0.56	0.83	0.81
ship12l	0.4	0.61	0.64	0.64	0.71	0.68	0.68	0.67	0.69	0.68
ship12s	0.4	0.67	0.74	0.73	0.85	0.91	0.91	0.74	0.80	0.79
sierra	0.4	0.71	0.84	0.83	0.82	0.89	0.88	0.86	0.92	0.92
stair	2.3	0.62	0.60	0.61	0.74	0.71	0.71	0.76	0.71	0.72
standata	0.8	0.38	0.71	0.68	0.47	0.78	0.75	0.42	0.77	0.74
standgub	0.7	0.30	0.71	0.67	0.30	0.71	0.67	0.33	0.73	0.69
standmps	0.7	0.36	0.65	0.63	0.37	0.66	0.63	0.40	0.66	0.64
stocfor1	3.6	0.47	0.54	0.53	0.64	0.65	0.65	0.66	0.66	0.66
stocfor2	0.2	0.74	0.80	0.79	0.73	0.73	0.73	0.74	0.73	0.73
tuff	2.6	0.34	0.37	0.36	0.39	0.39	0.39	0.36	0.37	0.37
vtp.base	2.3	0.58	0.58	0.58	0.70	0.66	0.67	0.72	0.67	0.68
wood1p	11.0	0.12	0.01	0.02	0.13	0.01	0.02	0.12	0.01	0.02
woodw	0.4	0.27	0.17	0.18	0.37	0.17	0.19	0.41	0.18	0.21

Table 5
μ Values for Partitions into Varying Numbers of Blocks

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
25fv47	0.97	0.88	0.65	0.73	0.69	0.58	0.45	0.27
80bau3b	0.78	0.62	0.60	0.59	0.56	0.52	0.43	0.41
adlittle	0.95	0.59	0.50	0.47	0.35	0.18	0.10	0.03
afiro	1.00	0.77	0.56	0.51	0.33	0.16	0.00	0.00
agg	0.80	0.74	0.65	0.44	0.27	0.19	0.19	0.18
agg2	1.00	0.89	0.71	0.76	0.59	0.53	0.48	0.45
agg3	1.00	1.00	0.83	0.78	0.66	0.57	0.53	0.50
bandm	0.91	0.77	0.71	0.66	0.58	0.43	0.34	0.34
beaconfd	0.73	0.49	0.48	0.46	0.44	0.44	0.38	0.28
blend	0.75	0.75	0.62	0.46	0.29	0.18	0.00	0.00
bnl1	0.84	0.75	0.69	0.62	0.58	0.52	0.48	0.37
bnl2	0.87	0.83	0.78	0.71	0.66	0.60	0.58	0.52
boeing1	1.00	0.73	0.69	0.66	0.62	0.57	0.44	0.34
boeing2	0.65	0.52	0.39	0.34	0.32	0.25	0.17	0.16
bore3d	0.85	0.74	0.65	0.62	0.52	0.49	0.41	0.38
brandy	0.80	0.66	0.57	0.49	0.42	0.40	0.34	0.31
capri	0.80	0.62	0.59	0.50	0.42	0.34	0.27	0.17
cycle	0.91	0.80	0.77	0.70	0.66	0.58	0.55	0.41
czprob	0.91	0.61	0.57	0.48	0.41	0.43	0.44	0.43
d2q06c	0.94	0.89	0.80	0.79	0.75	0.66	0.52	0.41
d6cube	0.32	0.06	0.04	0.04	0.02	0.00	0.00	0.00
degen2	0.71	0.61	0.50	0.45	0.37	0.29	0.21	0.13
degen3	0.78	0.60	0.54	0.45	0.34	0.30	0.25	0.20
dfl001	1.00	0.83	0.76	0.67	0.60	0.57	0.49	0.44
e226	0.81	0.71	0.63	0.59	0.46	0.39	0.35	0.34
etamacro	0.92	0.69	0.62	0.56	0.45	0.40	0.29	0.18
fffff800	0.83	0.54	0.45	0.34	0.33	0.28	0.28	0.27
finnis	1.00	0.87	0.80	0.69	0.66	0.61	0.56	0.45
fit1p	0.60	0.24	0.12	0.12	0.00	0.00	0.00	0.00

Table 6
μ Values for Partitions into Varying Numbers of Blocks (continued)

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
fit2p	0.14	0.04	0.02	0.02	0.02	0.00	0.00	0.00
forplan	0.96	0.88	0.84	0.68	0.62	0.54	0.42	0.35
ganges	1.00	0.92	0.88	0.78	0.74	0.66	0.57	0.48
gfrd-pnc	0.97	0.87	0.84	0.82	0.79	0.72	0.65	0.52
greenbea	0.96	0.83	0.73	0.61	0.57	0.47	0.38	0.26
greenbeb	1.00	0.90	0.79	0.66	0.62	0.51	0.41	0.28
grow15	0.97	0.72	0.68	0.60	0.55	0.00	0.00	0.00
grow22	0.66	0.64	0.61	0.52	0.39	0.18	0.00	0.00
grow7	0.76	0.53	0.49	0.30	0.00	0.00	0.00	0.00
israel	0.78	0.64	0.53	0.57	0.54	0.33	0.25	0.21
kb2	0.55	0.52	0.34	0.24	0.18	0.19	0.00	0.00
lotfi	0.92	0.65	0.61	0.57	0.49	0.52	0.47	0.40
maros	1.00	0.95	0.87	0.82	0.79	0.71	0.51	0.36
nesm	0.87	0.71	0.64	0.57	0.50	0.46	0.44	0.37
perold	1.00	0.83	0.73	0.63	0.55	0.36	0.28	0.21
pilot	0.75	0.62	0.51	0.37	0.23	0.21	0.15	0.14
pilot.ja	0.97	0.74	0.62	0.55	0.46	0.39	0.35	0.29
pilot.we	1.00	0.80	0.72	0.60	0.56	0.40	0.29	0.23
pilot4	1.00	0.99	0.77	0.60	0.49	0.44	0.39	0.21
pilot87	0.87	0.68	0.51	0.39	0.37	0.23	0.19	0.16
pilotnov	0.86	0.71	0.65	0.59	0.52	0.44	0.38	0.29
recipe	1.00	0.93	0.90	0.76	0.46	0.26	0.01	0.00
sc105	0.95	0.84	0.76	0.61	0.45	0.32	0.24	0.17
sc205	1.00	0.94	0.85	0.72	0.62	0.47	0.37	0.24
sc50a	1.00	0.88	0.78	0.58	0.41	0.25	0.00	0.00
sc50b	0.92	0.79	0.66	0.56	0.44	0.33	0.00	0.00
scagr25	0.97	0.91	0.86	0.85	0.75	0.69	0.62	0.55
scagr7	0.87	0.79	0.71	0.63	0.56	0.52	0.46	0.33
scfxm1	1.00	0.87	0.75	0.65	0.59	0.48	0.38	0.32
scfxm2	1.00	1.00	0.99	0.83	0.72	0.64	0.56	0.45
scfxm3	1.00	0.95	0.89	0.77	0.74	0.62	0.53	0.47

Table 7
μ Values for Partitions into Varying Numbers of Blocks (continued)

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
scorpion	1.00	1.00	0.97	0.94	0.92	0.87	0.66	0.45
scrs8	0.91	0.80	0.78	0.69	0.62	0.50	0.46	0.43
scsd1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
scsd6	0.25	0.64	0.17	0.04	0.02	0.04	0.00	0.00
scsd8	0.75	0.92	0.74	0.36	0.10	0.04	0.04	0.00
sctap1	0.90	0.81	0.76	0.62	0.55	0.46	0.38	0.15
sctap2	1.00	0.97	0.91	0.86	0.80	0.72	0.59	0.44
sctap3	1.00	0.94	0.89	0.83	0.77	0.71	0.61	0.53
seba	0.26	0.22	0.21	0.20	0.20	0.19	0.17	0.14
share1b	0.78	0.71	0.65	0.53	0.36	0.12	0.13	0.09
share2b	0.89	0.75	0.72	0.32	0.21	0.16	0.19	0.19
shell	0.65	0.62	0.60	0.53	0.47	0.44	0.41	0.36
ship04l	1.00	0.46	0.19	0.21	0.16	0.11	0.11	0.08
ship04s	0.56	0.47	0.42	0.41	0.38	0.36	0.35	0.34
ship08l	1.00	1.00	1.00	0.73	0.53	0.47	0.40	0.33
ship08s	0.88	0.89	0.82	0.76	0.73	0.70	0.66	0.64
ship12l	1.00	0.70	0.68	0.59	0.50	0.48	0.44	0.42
ship12s	1.00	0.98	0.91	0.90	0.87	0.85	0.79	0.79
sierra	1.00	1.00	0.88	0.79	0.75	0.77	0.69	0.52
stair	0.97	0.85	0.71	0.48	0.34	0.23	0.18	0.15
standata	0.90	0.77	0.75	0.68	0.66	0.59	0.57	0.54
standgub	0.87	0.69	0.67	0.63	0.62	0.58	0.53	0.52
standmps	0.86	0.76	0.63	0.56	0.52	0.43	0.46	0.43
stocfor1	1.00	0.75	0.65	0.56	0.49	0.43	0.30	0.13
stocfor2	0.90	0.80	0.73	0.73	0.69	0.63	0.55	0.44
tuff	0.78	0.43	0.39	0.32	0.30	0.31	0.27	0.20
vtp.base	0.95	0.80	0.67	0.62	0.53	0.49	0.49	0.36
wood1p	0.05	0.03	0.02	0.02	0.01	0.01	0.01	0.01
woodw	1.00	0.67	0.19	0.14	0.10	0.10	0.05	0.05