PARTITIONING MATHEMATICAL PROGRAMS FOR PARALLEL SOLUTION

by

Michael C. Ferris and Jeffrey D. Horn

Computer Sciences Technical Report #1232

May 1994

Partitioning Mathematical Programs for Parallel Solution[*]

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

May, 1994

Abstract

This paper describes heuristics for partitioning a general $M \times N$ matrix into doubly-bordered, block-diagonal form. Such heuristics are useful for decomposing large, constrained, optimization problems into forms that are amenable to parallel processing. The heuristics presented are all $O((M+N)^2 \log(M+N))$ and are easily implemented. 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.

1 Introduction

This paper describes several heuristics for partitioning a general $M \times N$ matrix into a doubly-bordered, block-diagonal form. Such a partitioning is important in many areas of numerical analysis where several partitioning heuristics exist for the special case of $N \times N$ symmetric matrices [5, 9]. We use our 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 such optimization problems into a manageable number of independent blocks of constraints, linked together by relatively few coupling variables and coupling constraints.

First, the doubly-bordered, block-diagonal form is described. 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 in terms of partitioning the graph of the matrix. The heuristics essentially have one degree of freedom associated with them which relates to the number of dummy nodes that are added to the associated graph. These dummy nodes enable the resulting blocks to have uneven sizes, and perhaps some blocks to be empty. Thus, by adding enough dummy nodes to the graph, we are able to accommodate problems that naturally split into less than the requested number of blocks.

[*]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.

[†]Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

In Section 3, we present some computational results to demonstrate the effectiveness of our heuristics. We give three sets of results to show well our partitioning algorithms perform *as partitioning algorithms.* That is, how close do they come to producing a doubly-bordered, block-diagonal matrix with the desired number of blocks. In the first set, 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 linking constraints and 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 this percentage of dummy nodes for the remainder of our computation. In the second set of results, we show how well our heuristic performs by taking a problem that is naturally doubly-bordered block-diagonal and randomly permuting its rows and columns. Our algorithm effectively reconstructs a doubly-bordered block diagonal form. These results should be useful in determining what class of problems is amenable to such partitioning and what the relative costs of treating the linking variables and constraints will be, as well as how balanced the computational load for each of the parallel processors will be.

The remainder of the paper shows one way to use the results of this algorithm in the solution of linear programs. Our approach is to remove the linking variables by replacing them with linking constraints and applying a dual method to these linking constraints. The dual problem is essentially a nonsmooth optimization problem which can be solved by an application of the bundle-level method. Using this approach, we illustrate the utility of partitioning matrices by decomposing the large constraint set of several linear programs from the NETLIB test suite into a reasonable number of independent constraint blocks and a relatively small number of coupling variables and constraints. We show how the objective function of such a linear program can be suitably modified so as to take advantage of the partitioning. Essentially, the resulting problem may be solved in parallel on as many processors as there are independent constraint blocks. The computational results that we give in Section 5 measures the *utility* of our partitioning algorithms in the efficient solution of large-scale, linear programs.

We note however, that 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 [6, 7, 26].

2 Matrix Partitioning Algorithms

Definition 1 *We shall call a matrix doubly-bordered, block-diagonal if it is of the following form:*

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

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*

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

a column-linking *or* column-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 *column-linking constraints* appear as *rows* in a doubly-bordered, block-diagonal matrix.

Definition 3 *We call each column of the submatrix*

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

a row-linking *or* row-coupling *constraint.*

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* or *coupled* by such a column. Again, the reader should note that *row-linking constraints* appear as *columns* in a doubly-bordered, block-diagonal matrix.

We note that p and q may take the value 0, in which case either the row-linking constraints or the column-linking 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 row-linking constraints can be removed by adding some columns to various blocks and extra column-linking constraints.

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:*

1. $V = R \cup C$, $R = \{r_1, r_2, \ldots, r_M\}$, $C = \{c_1, c_2, \ldots, c_N\}$.

2. $(r_i, c_j) \in E$ *if* $r_i \in R$, $c_j \in C$, *and* $a_{i,j} \neq 0$.

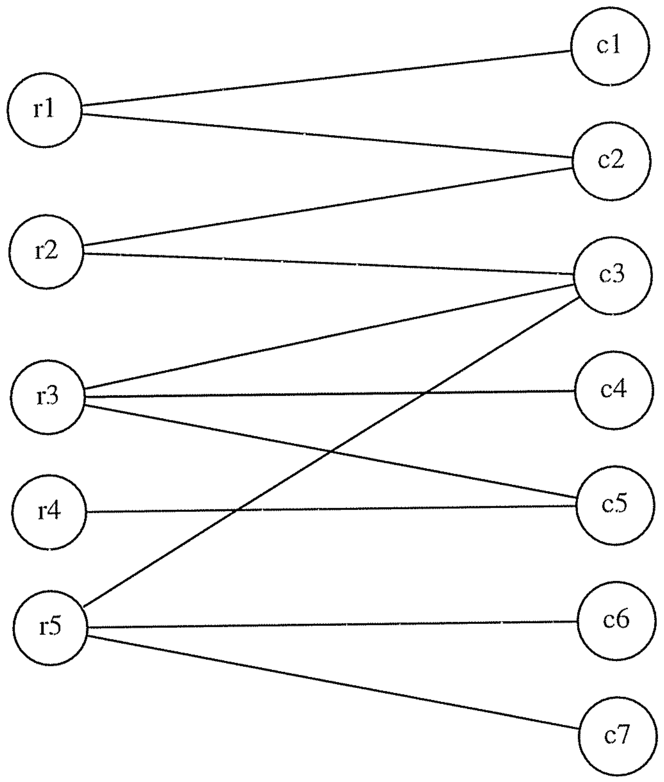

Figure 1: The associated bipartite graph G(A)

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 $G(A)$.

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 form the doubly-bordered block-diagonal matrix. 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.*

Kernighan and Lin give a highly effective heuristic for partitioning graphs so as to minimize the cost of the resulting partition [13]. We briefly describe their heuristics.

2.1 Two-Way Uniform Partitions

We first consider the problem of 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.

Let V be a set of $2n$ vertices of the graph $G = (V, E)$. 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.

An arbitrary partition A, B of V is chosen. 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.

2.2 Optimization Heuristic

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:

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

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.

2.3 Multiway Partitions

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. Clearly, this can be effectively implemented in parallel.

2.4 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.

2.5 Matrix Partitioning

We now discuss how the graph partitioning heuristics outlined above can be used to partition a matrix into doubly-bordered block diagonal 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 Kernighan-Lin procedure is applied to the resulting graph and a locally optimal graph partition is produced.

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.

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 doubly-bordered block-diagonal form.

It is relatively easy to transform a doubly-bordered block-diagonal form into a singly-bordered block-diagonal form. To accomplish this, we consider the variables corresponding to the row-coupling constraints

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

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 column-linking 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 [20]). Other techniques are described in [16]. 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 is C is sparse.

3 Partitioning Results

In this section, we present some computational results to demonstrate the effectiveness of the heuristics we have outlined above. We give three sets of results to show how well our partitioning algorithms perform *as partitioning algorithms.* That is, how close do they come to producing a doubly-bordered, block-diagonal matrix 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 [8]). 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

$$m^* := \max_{1 \le i \le K} m_i$$

$$n^* := \max_{1 \le j \le K} n_j$$

$$\alpha := \frac{1}{K^2} \sum_{i=1}^{K} \frac{m_i}{m^*} \sum_{j=1}^{K} \frac{n_j}{n^*}$$

$$\beta := \frac{\sum_{i=1}^{K} m_i \sum_{j=1}^{K} n_j}{MN}$$

We note that α is equal to one if each of the blocks have 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 percentage of the partition that is not part of the lower or right hand border. That means that $1 - \beta$ is the percentage 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 linking constraints and variables are penalized by adjusting the parameters p and q. Values of q near one (p near zero) will penalize linking constraints rather heavily, while values of q near zero (p near one) will tend to 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 would 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 1, Table 2 and Table 3. 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.

In most of the problems adding dummy nodes does not seem to help. The number of linking constraints is not significantly reduced by adding dummy nodes for many of the problems. That is to say, β does not increase very much as the number of dummy nodes is increased. In some cases, β actually decreases. This is because our heuristic only gives locally optimal partitions and adding dummy nodes to a problem can make the heuristic spend a good deal of its time shuffling dummy nodes, which can cause it to fall into a different (and less satisfactory) locally optimal partition. We also note that α, which measures the variability of block sizes, seems to be quite sensitive to increases in dummy nodes. 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 percent of dummy nodes is increased.

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 column-coupling constraints. This structure is then 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. This has a μ value of 0.94. The easier problem of finding 11 blocks results also finds a similar form without any difficulties. These results show that our algorithm effectively detects doubly-bordered block-diagonal form when it exists. We note that 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 5, this is a very small fraction of the time needed for solving the underlying linear program on the parallel machine. This

Problem	% Density	0 % dummy nodes			20 % dummy nodes			40 % dummy nodes		
		α	β	μ	α	β	μ	α	β	μ
25fv47	0.90	0.61	0.64	0.64	0.50	0.46	0.46	0.30	0.57	0.55
80bau3b	0.10	0.46	0.57	0.56	0.39	0.58	0.56	0.28	0.58	0.55
adlittle	8.40	0.52	0.47	0.47	0.34	0.40	0.39	0.24	0.48	0.45
afiro	9.80	0.41	0.47	0.46	0.29	0.50	0.48	0.30	0.51	0.49
agg	3.20	0.44	0.65	0.63	0.38	0.73	0.69	0.28	0.55	0.52
agg2	2.90	0.42	0.69	0.66	0.26	0.55	0.52	0.29	0.69	0.65
agg3	2.90	0.42	0.70	0.67	0.29	0.58	0.55	0.25	0.71	0.66
bandm	1.80	0.81	0.65	0.67	0.43	0.60	0.59	0.36	0.57	0.55
beaconfd	7.60	0.31	0.42	0.41	0.31	0.49	0.47	0.25	0.43	0.41
blend	8.40	0.55	0.56	0.56	0.33	0.54	0.52	0.34	0.59	0.56
bnl1	0.80	0.64	0.69	0.69	0.52	0.71	0.69	0.46	0.71	0.68
bnl2	0.20	0.64	0.78	0.76	0.54	0.78	0.75	0.35	0.59	0.57
boeing1	2.90	0.48	0.57	0.56	0.35	0.51	0.50	0.26	0.51	0.48
boeing2	6.60	0.31	0.37	0.37	0.28	0.42	0.41	0.29	0.51	0.48
bore3d	2.10	0.56	0.62	0.62	0.41	0.60	0.58	0.34	0.61	0.58
brandy	4.70	0.45	0.50	0.50	0.41	0.50	0.49	0.22	0.43	0.41
capri	1.90	0.49	0.58	0.57	0.50	0.61	0.60	0.32	0.54	0.51
cycle	0.40	0.66	0.75	0.74	0.49	0.68	0.66	0.44	0.66	0.64
czprob	0.40	0.24	0.49	0.46	0.22	0.42	0.40	0.20	0.33	0.32
d2q06c	0.30	0.68	0.72	0.72	0.44	0.61	0.59	0.32	0.56	0.54
d6cube	1.80	0.21	0.02	0.04	0.13	0.01	0.02	0.09	0.00	0.01
degen2	1.90	0.55	0.48	0.49	0.41	0.46	0.46	0.31	0.48	0.46
degen3	1.00	0.56	0.49	0.49	0.39	0.50	0.49	0.25	0.46	0.44
dfl001	0.10	0.81	0.58	0.60	0.56	0.42	0.43	0.43	0.42	0.42
e226	4.40	0.47	0.63	0.62	0.40	0.62	0.60	0.29	0.56	0.53
etamacro	0.90	0.71	0.53	0.55	0.60	0.50	0.51	0.41	0.54	0.53
fffff800	1.40	0.34	0.46	0.45	0.30	0.41	0.40	0.23	0.47	0.44
finnis	0.90	0.74	0.66	0.67	0.42	0.61	0.59	0.35	0.62	0.59
fit1d	56.30	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fit1p	1.00	0.17	0.11	0.12	0.07	0.00	0.01	0.00	0.00	0.00
fit2d	50.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fit2p	0.10	0.10	0.01	0.02	0.09	0.00	0.01	0.08	0.00	0.01
forplan	27.70	0.30	0.83	0.78	0.26	0.92	0.85	0.20	0.92	0.84
ganges	0.30	0.75	0.84	0.84	0.59	0.70	0.69	0.39	0.66	0.64
gfrd-pnc	0.50	0.72	0.83	0.82	0.57	0.81	0.78	0.39	0.81	0.77
greenbea	0.20	0.74	0.72	0.72	0.48	0.50	0.50	0.36	0.50	0.48
greenbeb	0.20	0.74	0.72	0.72	0.48	0.50	0.50	0.36	0.50	0.48

Table 1: 8 block partitions with varying percentages of dummy nodes

Problem	% Density	0 % dummy nodes			20 % dummy nodes			40 % dummy nodes		
		α	β	μ	α	β	μ	α	β	μ
grow15	2.90	0.77	0.65	0.66	0.47	0.52	0.52	0.29	0.52	0.50
grow22	2.00	0.54	0.56	0.56	0.51	0.59	0.58	0.39	0.47	0.46
grow7	6.20	0.50	0.46	0.47	0.42	0.47	0.47	0.21	0.35	0.33
israel	9.50	0.49	0.43	0.44	0.31	0.41	0.40	0.26	0.37	0.36
kb2	16.10	0.27	0.34	0.33	0.32	0.48	0.46	0.26	0.51	0.49
lotfi	2.30	0.59	0.56	0.56	0.35	0.54	0.52	0.25	0.53	0.50
maros	0.80	0.62	0.71	0.70	0.45	0.68	0.65	0.38	0.68	0.65
nesm	0.70	0.67	0.59	0.60	0.42	0.46	0.45	0.34	0.51	0.50
perold	0.70	0.74	0.60	0.61	0.45	0.52	0.52	0.37	0.52	0.51
pilot	0.80	0.42	0.47	0.47	0.54	0.49	0.50	0.25	0.49	0.47
pilot.ja	0.80	0.68	0.61	0.61	0.40	0.53	0.52	0.31	0.57	0.54
pilot.we	0.50	0.60	0.72	0.71	0.40	0.62	0.60	0.27	0.59	0.56
pilot4	1.30	0.74	0.61	0.62	0.48	0.57	0.56	0.43	0.63	0.61
pilot87	0.70	0.49	0.48	0.48	0.39	0.44	0.44	0.36	0.48	0.47
pilotnov	0.60	0.61	0.62	0.62	0.44	0.58	0.57	0.38	0.62	0.60
recipe	4.50	0.71	0.79	0.78	0.48	0.73	0.70	0.31	0.66	0.62
sc105	2.60	0.65	0.75	0.74	0.58	0.77	0.75	0.43	0.68	0.66
sc205	1.30	0.85	0.82	0.82	0.57	0.84	0.81	0.46	0.79	0.76
sc50a	5.50	0.58	0.65	0.64	0.53	0.70	0.68	0.50	0.65	0.63
sc50b	5.10	0.46	0.61	0.60	0.43	0.66	0.64	0.35	0.68	0.65
scagr25	0.90	0.79	0.84	0.84	0.66	0.73	0.72	0.41	0.73	0.70
scagr7	3.00	0.66	0.70	0.70	0.61	0.63	0.63	0.39	0.69	0.66
scfxm1	1.70	0.63	0.70	0.69	0.47	0.62	0.61	0.36	0.70	0.67
scfxm2	0.90	0.72	0.82	0.81	0.54	0.75	0.73	0.39	0.70	0.67
scfxm3	0.60	0.83	0.86	0.86	0.48	0.74	0.72	0.46	0.76	0.73
scorpion	1.20	0.85	0.86	0.86	0.58	0.80	0.77	0.49	0.78	0.75
scrs8	0.70	0.51	0.80	0.77	0.32	0.78	0.73	0.25	0.79	0.73
scsd1	5.30	0.00	0.00	0.00	0.13	0.10	0.11	0.00	0.00	0.00
scsd6	2.80	0.17	0.16	0.16	0.21	0.08	0.09	0.10	0.16	0.15
scsd8	1.00	0.78	0.71	0.72	0.45	0.58	0.57	0.40	0.45	0.45
sctap1	1.40	0.84	0.71	0.72	0.61	0.76	0.74	0.37	0.65	0.62
sctap2	0.40	0.88	0.80	0.81	0.61	0.78	0.76	0.39	0.67	0.64
sctap3	0.30	0.89	0.82	0.83	0.63	0.78	0.77	0.43	0.81	0.77
seba	0.90	0.17	0.20	0.20	0.20	0.32	0.31	0.14	0.35	0.33
share1b	4.50	0.46	0.65	0.63	0.34	0.71	0.68	0.36	0.76	0.72
share2b	9.50	0.71	0.71	0.71	0.64	0.77	0.76	0.28	0.55	0.52

Table 2: 8 block partitions with varying percentages of dummy nodes (continued)

Problem	% Density	0 % dummy nodes			20 % dummy nodes			40 % dummy nodes		
		α	β	μ	α	β	μ	α	β	μ
shell	0.50	0.54	0.59	0.59	0.43	0.60	0.58	0.30	0.56	0.53
ship04l	1.10	0.31	0.16	0.17	0.20	0.11	0.12	0.18	0.13	0.13
ship04s	1.10	0.32	0.41	0.40	0.28	0.41	0.40	0.35	0.40	0.40
ship08l	0.60	0.98	0.97	0.97	0.30	0.13	0.15	0.41	0.11	0.14
ship08s	0.60	0.53	0.82	0.79	0.48	0.56	0.56	0.33	0.59	0.56
ship12l	0.40	0.61	0.64	0.64	0.46	0.29	0.31	0.31	0.29	0.29
ship12s	0.40	0.67	0.74	0.73	0.34	0.64	0.61	0.36	0.68	0.65
sierra	0.40	0.71	0.84	0.83	0.51	0.72	0.70	0.44	0.83	0.79
stair	2.30	0.62	0.60	0.61	0.44	0.56	0.55	0.49	0.65	0.63
standata	0.80	0.38	0.71	0.68	0.22	0.70	0.65	0.17	0.70	0.64
standgub	0.70	0.30	0.71	0.67	0.26	0.67	0.63	0.17	0.72	0.67
standmps	0.70	0.36	0.65	0.63	0.29	0.55	0.53	0.21	0.56	0.52
stocfor1	3.60	0.47	0.54	0.53	0.40	0.51	0.50	0.29	0.51	0.49
stocfor2	0.20	0.84	0.90	0.89	0.56	0.75	0.73	0.37	0.75	0.71
tuff	2.60	0.34	0.37	0.36	0.24	0.37	0.35	0.19	0.39	0.37
vtp.base	2.30	0.58	0.58	0.58	0.35	0.60	0.58	0.27	0.64	0.60
wood1p	11.00	0.12	0.01	0.02	0.10	0.01	0.02	0.08	0.01	0.02
woodw	0.40	0.27	0.17	0.18	0.20	0.02	0.04	0.19	0.11	0.12

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

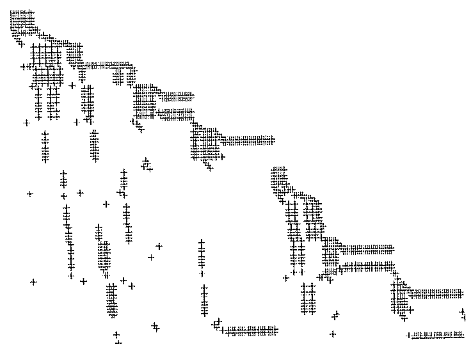

Figure 2: Share1b – Original Structure

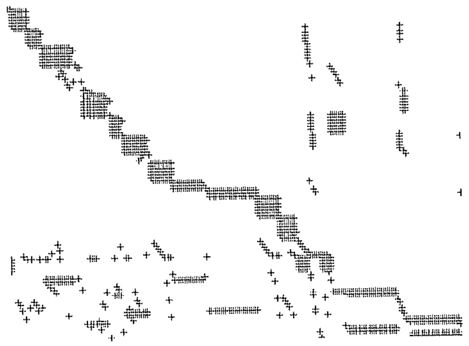

Figure 3: Share1b – 8 Block Partition, 0 % Dummy Variables ($\mu = 0.63$)

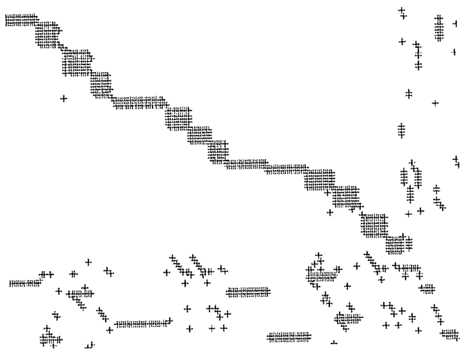

Figure 4: Share1b – 8 Block Partition, 20 % Dummy Variables ($\mu = 0.68$)

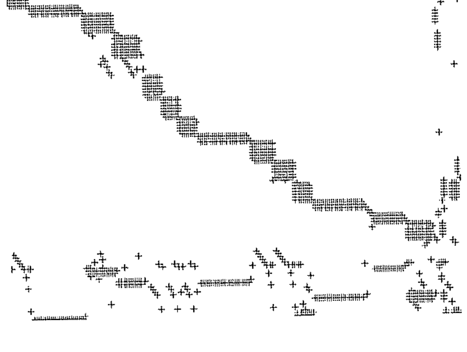

Figure 5: Share1b – 8 Block Partition, 40 % Dummy Variables ($\mu = 0.72$)

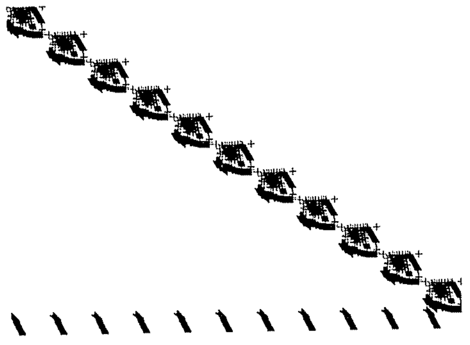

Figure 6: PDS Problem – Original Structure

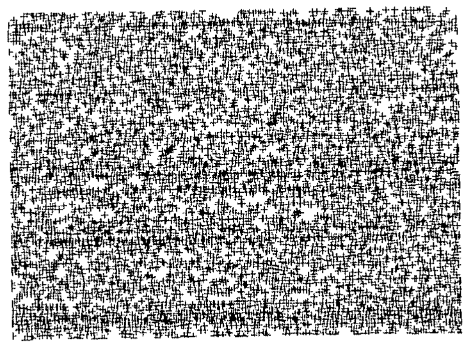

Figure 7: PDS Problem – Randomly Permuted

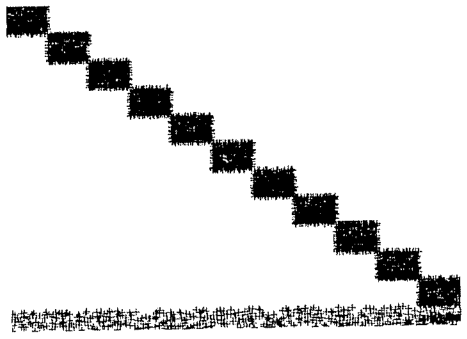

Figure 8: PDS Problem – Result of Partitioning ($\mu = 0.94$)

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 percentage 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 4, Table 5 and Table 6. Note that in all cases, the μ value generally decreases as the number of blocks increase as would be expected. There can be exceptions to this rule, however, since our heuristic may not always give a globally optimal solution to the partitioning problem. 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 linking constraints the tiebreaking could favor variables over constraints, etc).

Figure 9 through Figure 17 gives plots of how the resulting partitioned matrix appears for 20% of the variables as dummy for the problem 'stocfor2'. Note how effective the method appears to be in generating the blocks and the increase in coupling variables and constraints for more blocks. We note that 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.

4 Bundle-Level Decomposition

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 [8] 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. A detailed description of the form of the dual problem (due to Robinson [22, 23]) and the bundle method that we use for its solution (proposed in [14]) is the subject of the remainder of this section. Other related work on bundle methods can be found in [15, 27, 17, 2]. The final section of the paper gives some numerical results for our implementation.

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

$$\begin{array}{ll} \min\limits_{x=(x_1,\ldots,x_K)} & \sum\limits_{i=1}^{K} c_i^T x_i \\ \text{subject to} & B_i x_i = b_i, x_i \in X_i \\ & \sum\limits_{i=1}^{K} R_i x_i = r. \end{array}$$

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 [4, 17, 28, 11, 12]. We now describe the one that we shall use in this paper. For notational simplicity, we let

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

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
25fv47	0.69	0.62	0.46	0.52	0.49	0.41	0.32	0.19
80bau3b	0.73	0.58	0.56	0.55	0.52	0.49	0.40	0.38
adlittle	0.74	0.46	0.39	0.37	0.27	0.14	0.08	0.02
afiro	0.88	0.66	0.48	0.44	0.28	0.14	0.00	0.00
agg	0.85	0.79	0.69	0.47	0.29	0.20	0.20	0.19
agg2	0.83	0.65	0.52	0.56	0.43	0.39	0.35	0.33
agg3	0.81	0.69	0.55	0.52	0.44	0.38	0.35	0.33
bandm	0.76	0.64	0.59	0.55	0.48	0.36	0.28	0.28
beaconfd	0.71	0.48	0.47	0.45	0.43	0.43	0.37	0.27
blend	0.63	0.63	0.52	0.39	0.24	0.15	0.00	0.00
bnl1	0.84	0.75	0.69	0.62	0.58	0.52	0.48	0.37
bnl2	0.84	0.80	0.75	0.68	0.63	0.58	0.56	0.50
boeing1	0.84	0.53	0.50	0.48	0.45	0.41	0.32	0.25
boeing2	0.68	0.55	0.41	0.36	0.34	0.26	0.18	0.17
bore3d	0.76	0.66	0.58	0.55	0.46	0.44	0.37	0.34
brandy	0.69	0.57	0.49	0.42	0.36	0.34	0.29	0.27
capri	0.81	0.63	0.60	0.51	0.43	0.35	0.27	0.17
cycle	0.78	0.69	0.66	0.60	0.57	0.50	0.47	0.35
czprob	0.64	0.43	0.40	0.34	0.29	0.30	0.31	0.30
d2q06c	0.69	0.66	0.59	0.58	0.55	0.49	0.38	0.30
d6cube	0.16	0.03	0.02	0.02	0.01	0.00	0.00	0.00
degen2	0.65	0.56	0.46	0.41	0.34	0.27	0.19	0.12
degen3	0.71	0.54	0.49	0.41	0.31	0.27	0.23	0.18
dfl001	0.61	0.47	0.43	0.38	0.34	0.32	0.28	0.25
e226	0.77	0.68	0.60	0.56	0.44	0.37	0.33	0.32
etamacro	0.76	0.57	0.51	0.46	0.37	0.33	0.24	0.15
fffff800	0.74	0.48	0.40	0.30	0.29	0.25	0.25	0.24
finnis	0.84	0.64	0.59	0.51	0.49	0.45	0.41	0.33
fit1d	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fit1p	0.05	0.02	0.01	0.01	0.00	0.00	0.00	0.00
fit2d	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fit2p	0.07	0.02	0.01	0.01	0.01	0.00	0.00	0.00
forplan	0.97	0.89	0.85	0.69	0.63	0.55	0.43	0.35
ganges	0.83	0.72	0.69	0.61	0.58	0.52	0.45	0.38
gfrd-pnc	0.90	0.81	0.78	0.76	0.73	0.67	0.60	0.48
greenbea	0.66	0.57	0.50	0.42	0.39	0.32	0.26	0.18
greenbeb	0.66	0.57	0.50	0.42	0.39	0.32	0.26	0.18

Table 4: μ Values for Partitions into Varying Numbers of Blocks

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
grow15	0.74	0.55	0.52	0.46	0.42	0.00	0.00	0.00
grow22	0.63	0.61	0.58	0.49	0.37	0.17	0.00	0.00
grow7	0.73	0.51	0.47	0.29	0.00	0.00	0.00	0.00
israel	0.59	0.48	0.40	0.43	0.41	0.25	0.19	0.16
kb2	0.75	0.70	0.46	0.32	0.25	0.26	0.00	0.00
lotfi	0.78	0.55	0.52	0.49	0.42	0.44	0.40	0.34
maros	0.78	0.71	0.65	0.61	0.59	0.53	0.38	0.27
nesm	0.61	0.50	0.45	0.40	0.35	0.32	0.31	0.26
perold	0.77	0.59	0.52	0.45	0.39	0.26	0.20	0.15
pilot	0.74	0.61	0.50	0.36	0.23	0.21	0.15	0.14
pilot.ja	0.81	0.62	0.52	0.46	0.39	0.33	0.29	0.24
pilot.we	0.84	0.67	0.60	0.50	0.47	0.33	0.24	0.19
pilot4	0.80	0.72	0.56	0.44	0.36	0.32	0.28	0.15
pilot87	0.75	0.59	0.44	0.34	0.32	0.20	0.16	0.14
pilotnov	0.75	0.62	0.57	0.52	0.46	0.39	0.33	0.25
recipe	0.99	0.72	0.70	0.59	0.36	0.20	0.01	0.00
sc105	0.94	0.83	0.75	0.60	0.44	0.32	0.24	0.17
sc205	0.97	0.90	0.81	0.69	0.59	0.45	0.35	0.23
sc50a	0.88	0.77	0.68	0.51	0.36	0.22	0.00	0.00
sc50b	0.89	0.77	0.64	0.54	0.43	0.32	0.00	0.00
scagr25	0.81	0.76	0.72	0.71	0.63	0.58	0.52	0.46
scagr7	0.77	0.70	0.63	0.56	0.50	0.46	0.41	0.29
scfxm1	0.83	0.71	0.61	0.53	0.48	0.39	0.31	0.26
scfxm2	0.94	0.85	0.73	0.61	0.53	0.47	0.41	0.33
scfxm3	0.82	0.77	0.72	0.62	0.60	0.50	0.43	0.38
scorpion	0.87	0.80	0.77	0.75	0.73	0.69	0.52	0.36
scrs8	0.85	0.75	0.73	0.65	0.58	0.47	0.43	0.40
scsd1	0.61	0.18	0.11	0.07	0.04	0.00	0.00	0.00
scsd6	0.13	0.34	0.09	0.02	0.01	0.02	0.00	0.00
scsd8	0.58	0.71	0.57	0.28	0.08	0.03	0.03	0.00
sctap1	0.88	0.79	0.74	0.60	0.54	0.45	0.37	0.15
sctap2	0.87	0.81	0.76	0.72	0.67	0.60	0.49	0.37
sctap3	0.88	0.81	0.77	0.72	0.67	0.61	0.53	0.46
seba	0.38	0.32	0.31	0.30	0.30	0.28	0.25	0.21
share1b	0.82	0.74	0.68	0.55	0.38	0.13	0.14	0.09
share2b	0.94	0.79	0.76	0.34	0.22	0.17	0.20	0.20
shell	0.63	0.60	0.58	0.51	0.45	0.43	0.40	0.35

Table 5: μ Values for Partitions into Varying Numbers of Blocks (continued)

Problem	Number of Blocks Requested							
	2	4	8	16	32	64	128	256
ship04l	0.72	0.29	0.12	0.13	0.10	0.07	0.07	0.05
ship04s	0.53	0.45	0.40	0.39	0.36	0.34	0.33	0.32
ship08l	0.35	0.17	0.15	0.11	0.08	0.07	0.06	0.05
ship08s	0.60	0.61	0.56	0.52	0.50	0.48	0.45	0.44
ship12l	0.46	0.32	0.31	0.27	0.23	0.22	0.20	0.19
ship12s	0.71	0.66	0.61	0.60	0.58	0.57	0.53	0.53
sierra	0.90	0.84	0.70	0.63	0.60	0.61	0.55	0.41
stair	0.75	0.66	0.55	0.37	0.26	0.18	0.14	0.12
standata	0.78	0.67	0.65	0.59	0.57	0.51	0.49	0.47
standgub	0.82	0.65	0.63	0.59	0.58	0.55	0.50	0.49
standmps	0.72	0.64	0.53	0.47	0.44	0.36	0.39	0.36
stocfor1	0.85	0.58	0.50	0.43	0.38	0.33	0.23	0.10
stocfor2	0.90	0.80	0.73	0.73	0.69	0.62	0.55	0.44
tuff	0.70	0.39	0.35	0.29	0.27	0.28	0.24	0.18
vtp.base	0.82	0.69	0.58	0.54	0.46	0.42	0.42	0.31
wood1p	0.05	0.03	0.02	0.02	0.01	0.01	0.01	0.01
woodw	0.42	0.14	0.04	0.03	0.02	0.02	0.01	0.01

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

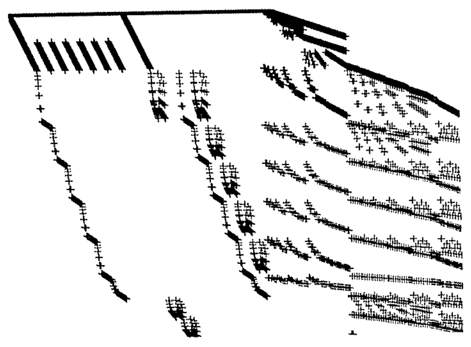

Figure 9: Stocfor2 – Original Structure

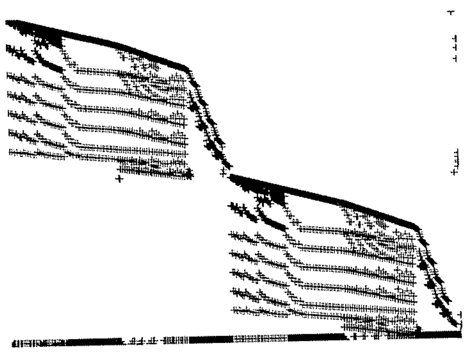
Figure 10: Stocfor2 – 2 Block Partition($\mu = 0.90$)

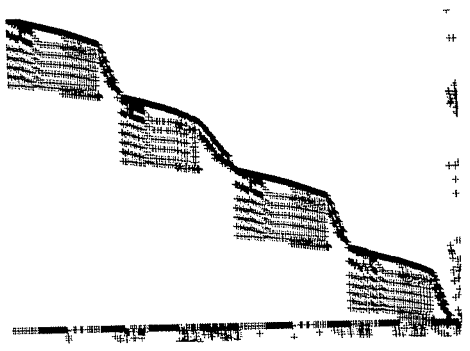
Figure 11: Stocfor2 – 4 Block Partition ($\mu = 0.80$)

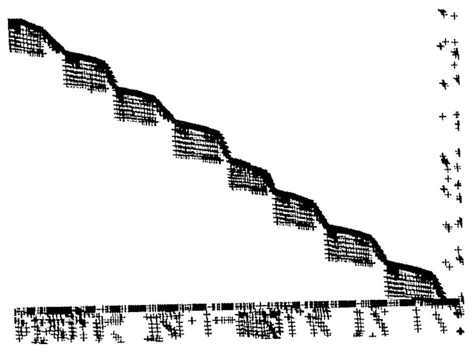
Figure 12: Stocfor2 – 8 Block Partition ($\mu = 0.73$)

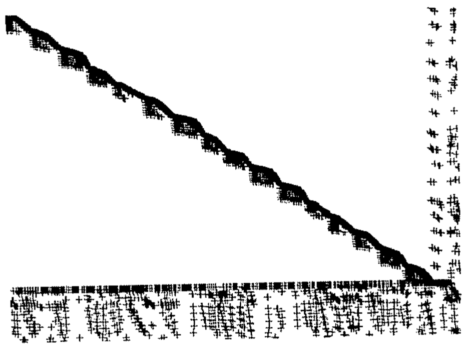

Figure 13: Stocfor2 – 16 Block Partition ($\mu = 0.73$)

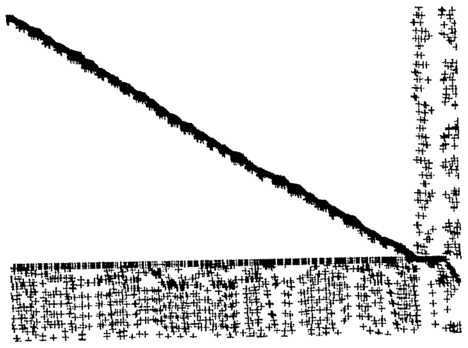

Figure 14: Stocfor2 – 32 Block Partition ($\mu = 0.69$)

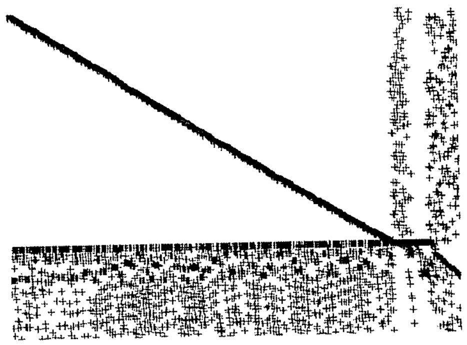

Figure 15: Stocfor2 – 64 Block Partition ($\mu = 0.63$)

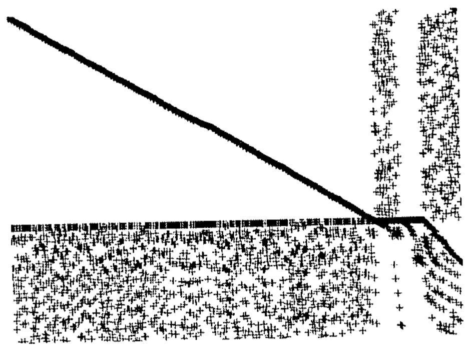

Figure 16: Stocfor2 – 128 Block Partition ($\mu = 0.55$)

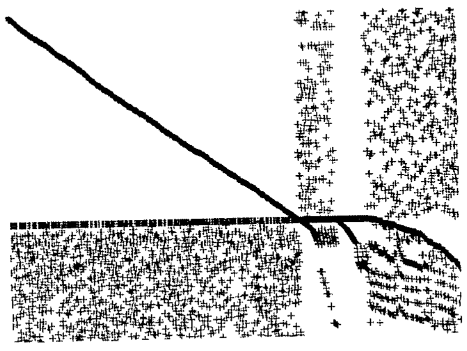

Figure 17: Stocfor2 – 256 Block Partition ($\mu = 0.44$)

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

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

Using a standard dualization technique [25], 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 column-linking 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 [24, 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 [14], which is now discussed in more detail.

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

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

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:

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

2. Evaluate f_*, f^* and x^* and let $\Delta = f^* - f_*$.

3. 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

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

4. 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 [14]) 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 linking constraints.

5 Parallel Solution of Linear Programs

The algorithm for solving linear programs given in the previous section has been implemented in Split-C [3] on the Thinking Machines CM-5 supercomputer and used to solve a variety of the largest linear programs in the NETLIB collection [8].

Split-C [3] is a parallel extension of the C programming language primarily intended for distributed memory multiprocessors and designed around two objectives. The first of these objectives is to capture certain useful elements of shared memory, message passing, and data parallel programming in a familiar context, while eliminating the primary deficiences of each paradigm. The second is to provide efficient access to the underlying machine, which in this work is a Thinking Machines CM-5. In our implementation, shared memory is used to handle data associated with the synchronization linear and quadratic programs. Split-C facilitates easy coding of the synchronization problem which obtains its data via message passing, while allowing the data for all the subproblems to be physically distributed across the processors. Much of this can also be carried out using CMMD [29], the message passing library of the CM-5. However, Split-C enables the code to be written in a more readily portable manner.

The first part of our code partitions the constraints of the problems according to the algorithm given in Section 2. This code is somewhat parallel in nature, but this has not been exploited in the work we present here. 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 a doubly-bordered block-diagonal form. In determining these permutations, we treat all the constraints as if they were equalities and do not add slack variables. The justification of this, is that it is very likely that the slack variables would be added 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 [21] and other computational enhancements [19]. 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 [18].

In Table 7 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. We note that our termination criterion was strict, requiring that two successive iterations have objective function evaluations within 10^{-9} of each other.

Note that the speedups for all of the problems are rather good. However, it should be noted that as the number of linking constraints grows, the efficiency decreases somewhat 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 regrettably beyond the scope of this work.

Problem	% Density	32 Block μ	Iterations	% Efficiency
sc205	1.30	0.59	8	87.3
scfxm3	0.60	0.60	9	91.1
sierra	0.40	0.60	11	88.9
scagr25	0.90	0.63	10	81.4
bnl2	0.20	0.63	8	88.1
sctap2	0.40	0.67	13	83.1
sctap3	0.30	0.67	12	83.7
stocfor2	0.22	0.69	12	80.2
scorpion	1.20	0.73	5	85.5
gfrd-pnc	0.50	0.73	8	86.8

Table 7: Parallel Solution Statistics

References

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

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

[3] D. Culler. *The Split-C Programming Language*. Computer Science Department, University of California, Berkeley.

[4] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[5] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*, chapter 8. Oxford University Press, New York, 1990.

[6] M. C. Ferris and O. L. Mangasarian. Parallel constraint distribution. *SIAM Journal on Optimization*, 1(4):487–500, 1991.

[7] M. C. Ferris and O. L. Mangasarian. Parallel variable distribution. Technical Report 1175, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1993. To appear in SIAM Journal on Optimization.

[8] D. M. Gay. Electronic mail distribution of linear programming test problems. *COAL Newsletter*, 13:10–12, 1985.

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

[10] J. Gilbert and E. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. *Xerox PARC Technical Report*, 1992.

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

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

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

[14] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. Rapports de Recherche 1508, INRIA–Rocquencourt, September 1991.

[15] 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, editors, *Nonlinear Programming*, volume 4, pages 245–282. Academic Press, 1981.

[16] D. Medhi. Decomposition of structured large–scale optimization problems and parallel optimization. Technical Report 718, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706, 1987. Ph.D. thesis.

[17] D. Medhi. Parallel bundle-based decompostion for large-scale structured mathematical programming problems. *Annals of Operations Research*, 22:101–127, 1990.

[18] R. Mifflin. A stable method for solving certain constrained least squares problems. *Mathematical Programming*, pages 141–158, 1979.

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

[20] S. S. Nielsen and S. A. Zenios. Solving linear stochastic network programs using massively parallel proximal algorithms. Technical Report 92-01-05, Decision Sciences Department, The Wharton School, University of Pennsylvania, Philadelphia, 1992.

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

[22] S. M. Robinson. Bundle-based decomposition: Description and preliminary results. In A Prékopa, J. Szelezcán, and B. Straazicky, editors, *System Modeling and Optimization*, volume 84 of *Lecture Notes in Control and Information Sciences*, pages 751–756. Springer-Verlag, Berlin, 1986.

[23] S. M. Robinson. Bundle-based decomposition: Conditions for convergence. In H. Attouch, J. P. Aubin, F. Clarke, and I. Ekeland, editors, *Analyse Non Linéaire*, pages 435–447. Gauthier-Villaars, Paris, 1989.

[24] R. T. Rockafellar. *Convex Analysis.* Princeton University Press, Princeton, NJ, 1970.

[25] R. T. Rockafellar. *Conjugate Duality and Optimization*, volume 16 of *Conference Board of the Mathematical Sciences*. SIAM, Philadelphia, PA, 1974.

[26] A. Ruszczynski. Regularized decomposition of stochastic programs: Algorithmic techniques and numerical results. WP 93-21, IIASA, A-2361 Laxenburg, Austria, 1993.

[27] 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*, 2:121–152, 1992.

[28] G. L. Schultz and R. R. Meyer. An interior point method for block angular optimization. *SIAM Journal on Optimization*, 1(4):583–602, 1991.

[29] Thinking Machines Corporation, Cambridge, MA. *CMMD Reference Manual, Version 3.0*, 1993.