ON THE COMPLEXITY OF PARTITIONING SPARSE MATRIX REPRESENTATIONS

Jóhann P. Malmquist
Ministry of Finance
Arnarhvoli
101 Reykjavik, Iceland

Edward L. Robertson†
Computer Science Department
Indiana University
Bloomington, In. 47405

† Supported by NSF Grant MCS - 8004337

ON THE COMPLEXITY OF PARTITIONING SPARSE MATRIX REPRESENTATIONS

J.P. MALMQUIST and E.L. ROBERTSON†

Abstract

A standard representation of a sparse matrix is a structure where non-zero elements are linked in rows and columns. A general graph structure corresponding to this representation is defined. The problem of partitioning such a graph into fixed size blocks, so that the number of inter-block links is minimized, is shown to be **NP**-complete.

Keywords: sparse matricies, graph partitioning, NP-completeness, data structures

1. Introduction

When a large matrix is sparse, with a high proportion of its entries zero (or some other fixed value), it is convenient to store only the non-zero entries of the matrix. The representation of such a sparse matrix is a doubly-linked structure [3, 7]. In this representation, each non-zero element belongs to two lists: a list of the non-zero elements of its column and of its row. Each list is ordered according to the appearance of the elements in the left-to-right or top-to-bottom traversal of the row or, respectively, column.

Figures 1 and 2 illustrate a sparse matrix (although it is not very sparse due to obvious space limitations) and its doubly linked representation.

27	12	0	39	0	0	0
0	71	46	0	43	0	0
10	0	0	95	0	0	47
0	47	0	0	0	11	0
0	0	13	0	0	0	38

Figure 1. A sparse matrix.

† Supported by NSF Grant MCS - 8004337

Links directed down and to the right; dashed lines represent row and column headers.

Figure 2. Doubly-linked representation of a sparse matrix.

This paper addresses the problem of partitioning the list representation of a sparse matrix in order to place the structure on secondary storage. Such partitioning arises only with extremely large matrices, often with matrices not directly derived from numerical problems. Indeed, the current research originated with the realization that a common data-base situation could be represented as a sparse matrix [6].

When any large data structure is placed on secondary storage, it must be partitioned into blocks, whose size is determined by the physical or logical constraints of the system. In most operating systems, blocks are determined by uniform partitions of the users address space and are "invisible" to the user, or at least largely beyond the users control. Our notion of blocking, however, includes the possibility of a more deliberate allocation of objects to blocks in order to meet various efficiency criteria. A primary consideration in evaluating the efficiency of a partition of a large data structure is the ways in which the structure is to be traversed. Thus a linear file is partitioned sequentially while a tree is partitioned attempting to keep each subtree on a block [5, 8]. (The only links between the objects in secondary storage are those explicitly present in the data structure, thus we are not concerned with such problems as building indices to files.) Since obtaining a block from secondary storage is relatively expensive, traversing an off-block link (a link from one block to another) is likely to be expensive and should therefore be minimized. This is facilitated by minimizing the number of off-block links.

Considering the data structure as a graph, the goal is to partition the nodes into fixed-size classes (blocks) so that the partition classes may be separated by cutting the minimal number of edges. The minimization of off-block links is of course conditioned by the probabilities that the various links be traversed. Thus, if it is known that a matrix will only be traversed row-wise, the column links can be ignored and the rows partitioned according to an efficient algorithm [1]. If row and column traversal are considered equally important, however, the problem of partitioning the sparse matrix representation is difficult, in a sense which has rapidly become a classic categorization for a difficult problem: the problem is **NP**-complete [2].

2. Formal Definition

Before analyzing the complexity of the partition problem, we give a formal definition of a sparse matrix graph and show that this corresponds to or intuitive notion of the doubly-linked representation of a sparse matrix. The goal of this brief digression is to clearly specify the problem we are considering and to allow more precise comparison with a large number of graph problems already known to be **NP**-complete.

DEFINITION. A *sparse matrix graph* is a connected directed acyclic graph in which the edges are labeled with two labels (we will use "R" and "C" , with edges called "R-edges" or "C-edges"). Each node has at most one in-arc and at most one out-arc of each label. Moreover, consider the partition such that two nodes are in the same R-class iff they are joined by a chain of R-edges (the *R-partition*). Then the R-classes must be strictly partially ordered by the C-edges. Correspondingly defined C-classes must also be strictly partially ordered by R-edges.

A doubly-linked representation of a sparse matrix clearly corresponds to a sparse matrix graph when the links are directed right or down across rows or, respectively, columns. To see that any graph satisfying the above abstract definition indeed corresponds to the doubly-linked list representation of some sparse matrix, consider an arbitrary sparse matrix graph and apply the following procedure:

> Refine each partial order into a linear order and use the R- and C-classes as indices for an appropriately dimensioned matrix. Each node is then placed in the position indexed by its R- and C-classes.

It is necessary to show that there is at most one entry in each R, C position. Because of the labelling restrictions, each C-class is linearly ordered by the C-edges. Thus, if two elements of some C-class also belonged to the same R-class, that R-class would be greater than itself in the C-ordering.

It is easy to see that the doubly-linked representation of the resulting matrix, with links directed toward increasing indices, is exactly the original graph.

The example in Figure 3 shows that the requirement that the R- and C-classes be ordered by C- and R-edges, respectively, cannot be omitted. Note that the row and column headers (shown dashed in Figure 2) are omitted from Figure 3 and all subsequent examples.

Having formally defined the special form of graph with which we are dealing, we now formally define the problem as well.

DEFINITION. An instance of the problem *sparse matrix graph partitioning* is a sparse matrix graph G = <V, E> (V = vertices, E = edges) and two positive integers p and k. Denote this instance by S<G, p, k>. The actual problem is to partition V into some number of disjoint subsets (blocks) such that each subset has at most p members and that at most k edges have endpoints in two distinct subsets.

Figure 3. An example of directed acyclic graph following label constraints which is not a sparse matrix graph.

3. Partition Complexity

The following proof will use a reduction to *graph partitioning*, a problem which is known to be **NP**-complete and which is in fact a generalization of the current problem [4, 2 problem ND14]. An instance of graph partitioning consists of G = <V, E>, p and k, denoted P<G, p, k> without any restriction on G. The problem is again to be solved by partitioning V into disjoint subsets with the same constraints.

THEOREM. *Sparse matrix graph partitioning is* **NP**-*complete.*

PROOF. It is easy to show the problem is in **NP**. Given a sparse matrix graph G = <V, E> and integers p and k, simply guess a partition of V. This partition is easily checked in polynomial time for the required conditions.

To show the problem is indeed complete is **NP**, consider an instance G = <V, E>, p, and k of the graph partition problem and construct an instance of the sparse matrix graph partition problem. Let G have maximum degree of d. Let q denote (p + 1)/2. For each node v_i in V = $\{v_1, ..., v_n\}$ construct a grid of q(d + l) horizontal by (d + 1) vertical elements. This is made a graph by adding edges directed rightward across rows and down columns. These edges are obviously labelled R and C respectively. We will refer to this matrix as the "node matrix" for v_i and to its nodes as "elements".

We now link the node matrices using the following procedure:

> Index the edges incident with node v_i from 1 to d (or less), such that if the index of edge (v_i, v_r) is less than the index of (v_i, v_s), then r<s. Let e be an edge in E connecting node i and node j together, where i<j. Assume e has index i′ for v_i and j′ for v_j. Connect the rightmost element in row i′ of the node matrix i to the leftmost element in row j′ of the node matrix i. Repeat the above process for every edge in the graph G. The resultant graph is denoted M_G.

This procedure produces a sparse matrix graph, as can be seen by the following: The interior elements of a node matrix obviously satisfy the degree and ordering constraints for a sparse matrix graph. The exterior elements of a node matrix are incident with two or three edges

within the node matrix and at most one is added, corresponding to some unique edge incident with the original vertex v_i. Since the edge matrices are linked in an order agreeing with the original ordering of V, each R-class is ordered properly.

Figures 4 and 5 illustrated the transformation for a simple graph. We use p = 1 and hence q = 1 to simplify the example. In Figure 5, distinct R-classes have been allocated distinct rows as in a conventional sparse matrix, but row and column headers are still omitted.

Figure 4. Graph example (d = 2).

Edges are directed right or down; headers are omitted.

Figure 5. Transformed example from Figure 4 (p = q = 1).

If P<G, p, k> is an instance of graph partition where G has the maximum degree of d, then P<M_G, pq(d + 1)^2, k> is an instance of the sparse matrix graph partitioning problem. We now claim that there is a suitable partition for P<M_G, pq(d + 1)^2, k> if and only if there is a partition for P<G, p, k>.

Suppose there is a partition of cost k for M_G with block size pq(d + 1)^2. By Lemma 6, which appears below, we know that there is a partition of cost at most k such that no node matrix is split between blocks. Moreover the maximum number of node matrices in a block

is p. But from the construction of M_G we know that node matrices are connected by an edge if and only if the corresponding nodes in G are connected by an edge. Therefore, there exists a valid partition for G, with block size p and cost k.

Now suppose that there is a partition of cost k for G, with block size p. Since each node of G corresponds to a $q(d+1)^2$ node matrix of M_G, the p or fewer node matrices corresponding to the nodes in a block in the partition of G will fit into a block of size $pq(d+1)^2$. Since no edges internal to a node matrix are cut by this process, the only cost is the k (or fewer) edges cut in the original partition of G.

The following lemmas are required to verify that partitioning of M_G directly corresponds to a partitioning of G. They all are done in the context of partitioning M_G with block size $pq(d+1)^2$, where the graph G has degree d.

LEMMA 1. *If a node matrix is split between blocks, with more than d internal node matrix edges crossing a block boundary, then a partition of lower cost is obtainable by moving the entire node matrix to a free block.*

PROOF. There are at most d external edges connected to the node matrix. □

LEMMA 2. *If a node matrix is split over a block boundary for an optimal partition, then more than p/(p + 1) of the node matrix elements must be in the same block. Call this block the home block for the node matrix.*

PROOF. Since it is not possible to divide a node matrix horizontally or vertically with d cuts, the best separation is to cut off a corner with perimeter d. This will separate $(d/2)^2$ out of $q(d+1)^2$ elements, or fewer than $1/(p+1)$ the total number. □

LEMMA 3. *In an optimal partition, no block is the home block for more than p + 1 node matrices.*

PROOF. Say some block is home for r node matrices. Then $rq(d+1)^2p/(p+1)$ is the least number of nodes occupied by these node matrices and this quantity may not exceed the block size, $pq(d+1)^2$. □

LEMMA 4. *If a node matrix is split over a block boundary for some partition and if the home block of the node matrix has sufficient space, then "bringing home" the remainder of the split node matrix would result in a partition of lower cost.*

PROOF. In order to split a node matrix, at least one more internal edge must be cut than the number of external edges saved. □

LEMMA 5. *In an optimal partition, no block is the home block for more than p node matrices.*

PROOF. Because only p complete node matrices can fit onto a single block, space for an extra node matrix would have to be made available by omitting small portions of node matrices

from the block. Say m node matrices have portions placed on other blocks. We refer each of these as node matrix i, for some unique i between 1 and m. Since this is essentially a problem of maximizing area for a given perimeter, the best way to do this within the given constraints is to omit a rectangular portion from a corner each of the m node matrices. Say the corner omitted from node matrix i has height y_i and width x_i. Then the following constraints hold

1. $x_i + y_i \leq d$
2. $\Sigma x_i y_i \geq q(d+1)^2$
3. $\Sigma x_i \leq d$

Constraint 1 is simply Lemma 1 and constraint 2 is the requirement that the total area saved is the size of a node matrix. Constraint 3 follows because the total extra cost of the omissions cannot exceed d, since otherwise it would be more economical to move one node matrix to a free block and "bring home" the rest of the omitted portions (by Lemma 3, only one node matrix need be so displaced and moving one node matrix will allow the others to be "brought home"). Although the y_i costs could be balanced by savings in links between node matrices, the x_i costs can never by made up.

However, these three constraints cannot all be satisfiable, since

$$\Sigma x_i y_i \leq \Sigma x_i(d - x_i) = d \Sigma x_i + \Sigma x_i^{\,2} \leq d^2 < (d+1)^2 \leq q(d+1)^2$$

follows from 1 and 3, contradicting 2. □

LEMMA 6. *Given any partition of M_G, there is one of equal or lower cost with every node matrix on a single block. Hence, in an optimal partition no node matrix is split over a block boundary.*

PROOF. If no node matrix is split, we are done. Hence, assume that one or more node matrices are split over a block boundary. From Lemmas 2 and 3, we can find a partition of lower cost where each node matrix has a home block, and each block is home for at most p node matrices, if this situation does not already hold. Finally, the cost may be reduced by "bringing home" all remaining portions of node matrices not on their home blocks, as in Lemma 4. Since this lowers the cost of each node matrix involved, it obviously reduces the cost of the entire partition. □

This completes the proof of the theorem.

4. Extensions and Conclusion

The above construction was done with several restrictions, but these were mainly to simplify the proof and can be removed by a more elaborate construction.

The first elaboration is to add row and column headers to the sparse matrix graph. The implementation of headers which requires their inclusion in the partitioning scheme forms linear lists of row and column headers. This corresponds to adding an additional row and column at the top and left of the matrix, with no zero entries in either. If single node headers were added to the above construction, the partitioning arguments would no longer hold. However, a broad "band" of nodes, both headers and regular entries, can be added to the top and left of the above construction. These new nodes would occupy many more rows and columns than the original construction, and hence there would be low relative density of links external to the new nodes versus internal links. Thus, the partition of the added nodes is essentially forced in an optimal partition.

The above remarks, if implemented, could greatly decrease the sparsity of the associated matrix. However, matrices may be made arbitrary sparse (measured as non-zero elements over total elements) by adding new rows and columns which are zero except on the diagonal.

A final variation for which partitioning is still **NP**-complete is when row links have a different cost than column links, but both are non-zero (as opposed to the case when one cost is zero as discussed above). When the difference of costs is large (a factor of six or greater remarks), the whole construction becomes simple. When the costs are close together, "rotate" the entire construction if necessary, so that the links of higher cost correspond to the columns in the construction. This has the effect, in particular, of strengthening condition (3) of Lemma 5.

Now we must at last address the significance of a proof of **NP**-completeness. All that has been shown is the worst-case behavior of the general problem. There is no implication about how often the worst cases occur, about average behavior or about a host of other practical concerns. What proving **NP**-completeness does do is direct our attention to heuristics and approximation algorithms for the problem on hand. Indeed, the authors have been investigating linear-time heuristics for partitioning sparse matrix representations [6], but it is not necessary to limit investigations with so strong a constraint.

REFERENCES

(1) B.W. Kernighan, *Optimal Sequential Partitions of Graphs,* Journal of ACM, 18. 1. (January 1979). pp. 34-40.

(2) M.R. Garey and D.S. Johnson, *Computers, Complexity and Intractability: A Guide to the Theory of NP-completeness,* (1979), H.P. Freeman and Sons, San Francisco, California.

(3) E. Horowitz and S. Sahni, *Fundamentals of Data Structures,* (1976), Computer Science Press, Potomac, Maryland.

(4) L. Hyafil and R.L. Rivest, *Graph Partitioning and Constructing Optimal Decision-Trees are Polynomial Complete Problems,* IRIA report 33, (October 1973).

(5) J.A. Lukes, *Efficient Algorithm for the Partitioning of Trees,* IBM Journal of Research and Development, 18, 3, (May 1974), pp. 217-224.

(6) J.P. Malmquist, *Storage Allocation for Access Path Minimization in Network-Structured Data Bases,* Ph.D. Thesis, The Pennsylvania State University, (May 1979), University Park, Pennsylvania.

(7) D.E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms,* Addison-Wesley, Reading, Massachusetts.

(8) M. Schkolnick, *A Clustering Algorithm for Hierarchical Structures,* ACM Transactions on Database Systems, 2, 1, (March 1977), pp. 27-44.

BUREAU OF THE BUDGET
ARNARHVOLI
101 REYKJAVIK
ICELAND

AND

IBM T.J. WATSON RESEARCH CENTER
YORKTOWN HEIGHTS, NEW YORK 10598
USA

COMPUTER SCIENCE DEPARTMENT
INDIANA UNIVERSITY
101 LINDLEY HALL
BLOOMINGTON, INDIANA 47405
USA