SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305

A COMBINATORIAL APPROACH TO SOME SPARSE MATRIX PROBLEMS

by

S. Thomas McCormick

TECHNICAL REPORT SOL 83-5

June 1983

Research and reproduction of this report were partially supported by National Science Foundation Grants MCS-8119774, MCS-7926009 and ECS-8012974; the Department of Energy Contract DE-AM03-76SF00326, PA# DE-AT03-76ER72018; Office of Naval Research Contract N00014-75-C-0267; Army Research Office Contract DAAG29-81-K-156.

Table of Contents

List of Tables

Chapter 1

Introduction

1.1. Introduction

Mathematical modeling is a popular tool used to predict or control an uncertain future. The complexities of a real-world system are abstracted into variables linked by mathematical relationships that are hoped to capture the essential behavior of the system.

Some models are merely **descriptive**, that is, they attempt only to predict the course of events based on a set of assumptions. But many models incorporate control variables that can be adjusted to try to affect the future. Such models are **prescriptive** rather than descriptive.

A paradigm from economics that is often used in prescriptive models is the assumption that the controls are adjusted so as to maximize a utility function (or minimize a cost, which is effectively a negative utility). This point of view makes many prescriptive models into **constrained optimization** models, which are of the form

$$\begin{aligned} &\min f(x) \\ &\text{subject to} \quad c(x) = 0 \end{aligned} \tag{1.1.1}$$

where x represents the control variables, $f(x)$ is a "cost" that is to be minimized, and $c(x) = 0$ represents the constraints imposed on the variables of the model by the structure of the system under study.

Since models of the general form (1.1.1) have become widely used, methods for solving them have been intensively studied. A good general reference for practical modern methods of solution is Gill, Murray and Wright (1981).

As the capacity and speed of computers have dramatically risen, our ability to solve larger and larger models has also increased. But not all advances in methods are attributable to the existence of faster machines. Since the time that the first optimization models were introduced, particularly after the advent of linear programming in 1947 (see Dantzig (1963)), it has been recognized that real-world problems are usually specially structured. Clever algorithms can take advantage of special structures and speed up the solution of optimization models. The increases in size and solution speed greatly exceed the improvements in computer hardware.

One of the earliest and most important kinds of special structure that was recognized was **sparsity**. Roughly speaking, sparsity means that a variable interacts directly with only a few other variables. Thus in (1.1.1), sparsity would mean that most of the entries of the Jacobian of c are zero *for all* x. Sparsity tends to appear naturally because most variables in an optimization model interact only with other variables that are fairly "close" in either space or time. A classic example is a transportation network, where nodes of the network interact only with their immediate neighbors.

Sparsity is exploited in two ways by solution methods. First, since most of the potential information is zero, the number of pieces of data that must be stored in the computer is drastically reduced. Second, operations on sparse data can be done faster by exploiting sparsity. For example, in multiplying two sparse matrices, products involving a zero need not be computed.

The importance of sparse methods is evidenced in many ways. Every commercial linear programming code uses sparse matrix methods in the representation of the problem and for handling its bases. The most recent Sparse Matrix Symposium attracted 119 researchers who listened to 60 presented papers. A keyword search of the on-line Math Reviews on MATHFILE turned up 413 reviewed papers on sparsity since 1972.

This thesis continues the trend of research into better ways of exploiting sparsity by considering two different problems in which sparsity is crucial.

The first problem, discussed in Chapter 2, is to compute a finite-difference approximation of a sparse Hessian matrix with a minimum number of gradient evaluations. For many functions, sparsity allows a

clever method to approximate the Hessian with surprisingly few gradient evaluations. Some of the results in Chapter 2 appeared in McCormick (1981).

The second problem, discussed in Chapter 3, is that of making a given sparse matrix even sparser, perhaps as sparse as possible. The motivation is that increased sparsity should lead to savings in time and storage. An earlier version of some of the material in Chapter 3 appeared in Hoffman and McCormick (1982).

This technical report is a reformatted version of McCormick (1983).

1.2. Background and Notation

Though the problems studied in this thesis appear to be numerical by nature, they are both amenable to a combinatorial approach. Much of the preliminary material in Chapters 2 and 3 is aimed at showing the underlying combinatorial nature of the problems. The presentation assumes a basic knowledge of combinatorics, though references are given as new concepts are encountered. A basic reference for graph theory is Bondy and Murty (1976). Two good references for combinatorial optimization, and for bipartite matching in particular, are Lawler (1976) and Papadimitriou and Stieglitz (1982).

An important combinatorial concept that appears constantly in both chapters is the notion of **computational complexity**, which attempts to determine whether some problems are inherently more difficult to solve than others. The most important tool available for this purpose is **NP-Completeness**. Roughly speaking, a problem X is NP-Complete if it is as hard to solve as any of the well-known hard combinatorial problems such as the Traveling Salesman Problem or the Graph Coloring Problem. A good reference for NP-Completeness is Garey and Johnson (1979).

A problem X is shown to be NP-Complete by **reducing** a known NP-Complete problem Y to X. Reducing Y to X means that a polynomial way of encoding an instance of Y into an instance of X is demonstrated that has the property that solving the instance of X also solves the encoded instance of Y. Thus a **fast** (polynomial time) algorithm for solving X would also solve Y in polynomial time, so that X can be no easier than Y.

Strictly speaking, it is not known whether NP-Complete problems are actually harder than problems that have known polynomial algorithms, but the conventional wisdom among complexity theorists is that any algorithm that solves such a problem must take an exponential number of steps, so that NP-Completeness is tantamount to practical intractability. This belief should not be taken to imply that there is no hope of ever making any progress on an NP-Complete problem. One of the most active areas of complexity research is in finding and analyzing **heuristic** algorithms for NP-Complete problems (algorithms that only approximately solve a problem, or efficiently solve a subset of instances).

On occasions we shall want to distinguish between typical problems encountered in practice and arbitrarily structured problems. It is a truism of sparsity research that many practical problems have ill-defined additional structure that tends to make them more tractable than, say, randomly generated problems. When we want to refer to this phenomenon, we shall write of "real" or "real-life" or "practical" problems.

Despite attempting to fully "combinatorialize" our problems, at some points their numerical properties have to be considered. The reader should be aware that finite-precision arithmetic (a model of computer arithmetic) differs in many respects from "exact" arithmetic. The issues involved in trying to maintain accuracy in numerical computations are encompassed by the terms **numerical stability** and **conditioning**. An introduction to this subject is given in Dahlquist and Björck (1974).

Our notational conventions are as follows. A term is printed in **bold-face** when it is being defined, and *slanted* type is used for emphasis. Capital letters A, B, ..., are used for matrices and index sets, small letters a, b, ..., for vectors and scalars, and script letters $\mathcal{D}$, $\mathcal{G}$, ..., for graphs and matchings. Theorems, equations and tables are all referred to by a three part number of the form $x.y.z$, which means the z^{th} occurrence of that object in major section $x.y$. The end of a proof is marked by the symbol "□".

Chapter 2

Approximating Sparse Hessians

2.1. Introduction to Sparse Hessians

In numerical optimization procedures it is sometimes necessary to evaluate the Hessian matrix

$$H(x^0) = \left(\frac{\partial^2 F(x^0)}{\partial x_i \partial x_j} \right)$$

of a function $F: \mathrm{R}^n \rightarrow \mathrm{R}$. It is usually preferable to evaluate $H(x^0)$ analytically, but it is not always possible to do so. For instance, F may not be known analytically (if, say, F is the output of a simulation), F may be of a form that makes H very complicated to evaluate analytically, or the user of an optimization routine may simply be unwilling to provide an evaluation routine for H. These considerations make it useful for the designer of a "black box" optimization routine to include a facility for approximating H by finite-differencing.

We shall assume that there is a way to evaluate the gradient of F, call it $g(x) = (\frac{\partial F(x)}{\partial x_1}, \ldots, \frac{\partial F(x)}{\partial x_n})$, for use in finite-differencing. The fundamental fact that is used in finite-differencing is that if d is a "suitably" small (see Gill, Murray and Wright (1981), Sections 4.6.1 and 8.6, for a discussion of difference interval sizes) perturbation vector, then differencing g along direction d gives the linear equations

$$d^T \hat{H}(x^0) = g(x^0 + d) - g(x^0), \tag{2.1.1}$$

where $\hat{H}(x^0)$ is an approximation to $H(x^0)$. Note that the right hand side of (2.1.1) is calculated from the gradient evaluation routine, the "unknowns" are the entries of $\hat{H}$, and (2.1.1) is a system of n equations, one for each component of the gradient.

The most common and straightforward method of approximating $H(x^0)$ is successively to choose d in (2.1.1) to be a small multiple of each of the unit vectors $e^1, e^2, \ldots, e^n$. Let δ_i be the chosen difference interval for the i^{th} coordinate. When $d = \delta_i e^i$ the j^{th} equation in (2.1.1) is

$$\delta_i \hat{h}_{ij}(x^0) = g_j(x^0 + \delta_i e^i) - g_j(x^0),$$

thus allowing us to solve for an approximation to all of row i of $H(x^0)$. For any smooth F, H is symmetric, and so the $\hat{H}$ resulting from the usual method is symmetrized by setting

$$\hat{H} \leftarrow \frac{1}{2}(\hat{H} + \hat{H}^T).$$

Though equations (2.1.1) are trivial to solve when unit vectors are used for differencing, the procedure has one great drawback. When considering the running times of optimization routines, the standard assumption is that calculating $g(x)$ is expensive relative to other operations. The value $g(x^0)$ must be calculated by an optimization routine for other purposes, so we do not include it in evaluation counts. In addition to $g(x^0)$, the usual procedure requires n gradient evaluations for each approximation of H, and so can be prohibitively expensive even for moderately large n. As is shown later in Section 2.5, when the Hessian has no special structure the number of gradient evaluations cannot be reduced below n, making explicit approximation of H through finite-differencing unattractive. In some contexts, adequate approximations to H can be obtained efficiently through other means, see, *e.g.*, the vast literature on quasi-Newton methods (Gill, Murray and Wright (1981), Section 4.5.2). However, even with such methods, an explicit Hessian approximation can be useful for distinguishing between a saddle point and a true minimum.

The saving grace is that H often has special structure that can be exploited. Sometimes it is known from the structure of the problem that $h_{ij}(x) = 0$ for some i and j, *independent of* x. A Hessian is said to be **sparse** when such information is known about a large proportion of its entries. It is convenient to represent the sparsity information by a matrix of $\times$'s and 0's, where "0" represents an entry known to be zero for all x of interest, and "$\times$" represents any other entry. Such a matrix is called the **sparsity pattern** of H, and inherits its symmetry. For example, if $F(x, y, z) = x^3 + (y + z)^3$, the sparsity pattern of its Hessian is

$$\begin{pmatrix} \times & 0 & 0 \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix}.$$

As a slight generalization, note that it may also be known that an $h_{ij}(x)$ is a non-zero constant independent of x. Such an entry can be treated almost like a zero in this context, the only difference being that the constant must be subtracted from the right hand side of (2.1.1) at the appropriate point. For simplicity, we shall subsequently consider only the zero/non-zero distinction, though only minor changes are needed to adapt the results to the constant case as well.

As an example of how sparsity can be used to approximate a Hessian more efficiently, consider the "arrowhead" sparsity pattern (see Powell and Toint (1979), p. 1060):

$$\begin{pmatrix} \times & 0 & 0 & \cdots & \times \\ 0 & \times & 0 & & \times \\ 0 & 0 & \times & & \times \\ \vdots & & & \ddots & \vdots \\ \times & \times & \times & \cdots & \times \end{pmatrix}. \tag{2.1.2}$$

Choose the first difference direction to be $d^1 = \delta_n e^n$. The resulting j^{th} equation in (2.1.1) is

$$\delta_n \hat{h}_{jn}(x^0) = g_j(x^0 + d^1) - g_j(x^0).$$

These equations can be solved for $\hat{h}_{jn}(x^0)$, $j = 1, 2, \ldots, n$. Choose the second difference direction to be $d^2 = \sum_{i=1}^{n-1} \delta_i e^i$. The j^{th} equation of (2.1.1) is now

$$\delta_j \hat{h}_{jj}(x^0) = g_j(x + d^2) - g_j(x^0), \quad j = 1, 2, \ldots, n-1,$$

yielding the remaining non-zeros in H. Thus the n evaluations necessary in the usual method have been reduced to just two by using special difference directions. With the assumption that gradient evaluation is expensive, this is a significant saving and makes finite-difference approximations feasible for large-scale optimization.

Suppose further that H has the following truncated arrowhead structure:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 & \times \\ 0 & 0 & \times & 0 & \times \\ 0 & 0 & 0 & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}.$$

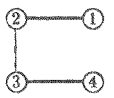
Note that the first, second and fifth equations of (2.1.1) are not used in the first evaluation. If g can be evaluated component by component, more time could be saved by evaluating only the third and fourth components of $g(x^0 + d^1)$. But it often happens that the components of g have common subexpressions that make evaluating one component of g nearly as expensive as evaluating all of g, causing these apparent savings to vanish. Also, many times g is available only as a user-written black box, and so it is not possible to specify that only a subset of components be evaluated. These considerations lead us to assume henceforth that g can be evaluated only as a whole, not component by component. However, it is easy to see how to adapt

the output of many of the available heuristics so that one can take advantage of component by component evaluation when it is available.

Our general goal now is to approximate the Hessian of a given function F when H has a known sparsity pattern, using the minimum possible number of gradient evaluations. This problem has been previously considered by several authors, including Powell and Toint (1979), Coleman and Moré (1982) and Thapa (1980), Section 5.1. In order for an approximation method to be practical it must be fast and it must be numerically stable. Searching for a balance between these two competing goals has led previous researchers to consider the problem under various restrictions on the form of equations (2.1.1). A systematic way of classifying these restrictions is presented in Section 2.2, and a subclassification of the so-called direct methods is presented in Section 2.3.

In order to better understand the examples in Sections 2.2 and 2.3 and to be able to analyze the complexity of different classes of methods, it is helpful to have a graph representation of sparsity patterns. Since the sparsity pattern of H is symmetric, a natural model to choose is the graph $\mathcal{G}(H)$ with node set $N = \{1, 2, \ldots, n\}$ and edges $E = \{\{i, j\} \mid h_{ij} \text{is not known to be } 0\}$. Thus the symmetry of H corresponds to the undirectedness of $\mathcal{G}(H)$. When drawing pictures of $\mathcal{G}(H)$, loops that stem from non-zero diagonal entries in H will be suppressed. For example, the following sparsity pattern corresponds to the displayed graph:

$$\begin{pmatrix} \times & \times & 0 & 0 \\ \times & \times & \times & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} \leftrightarrow$$



Conversely, any undirected graph (possibly with loops) clearly corresponds to the sparsity pattern of some matrix H.

Using this graph model, it is proved in Section 2.4 that all the variations of the direct methods considered in Section 2.3 are NP-Complete. Section 2.4 concludes with some positive results about heuristics for direct methods to counterbalance the negative complexity results.

In order to be able to quantify the performance of heuristics, it would be useful to have an easily computable lower bound on the number of evaluations needed. The number of evaluations needed when *no* restrictions are placed on the difference directions is clearly a lower bound. Some results about this bound and how to compute it efficiently are presented in Section 2.5.

This chapter of the thesis concludes with Section 2.6, which points out the unresolved questions in the preceding sections and suggests areas for future research.

2.2. Classifying Approximation Methods

Denote $(g(x^0 + d^l) - g(x^0))^T$ by Δ^l and an approximation to $H(x)$ by $\hat{H}(x)$. An **approximation method** is an algorithm that, when given F and the sparsity pattern of its Hessian, chooses fixed (independent of x) difference directions $d^1, d^2, \ldots, d^k$ so that the nk equations

$$\hat{H}(x^0)d^l = \Delta^l, \quad l = 1, 2, \ldots, k, \tag{2.2.1}$$

(which are just (2.1.1) re-written in the new notation) have a subsystem that can be uniquely solved to yield $\hat{H}(x^0)$. When the i, j entry of the sparsity pattern is zero, $\hat{h}_{ij}$ ($= \hat{h}_{ji}$) is set to zero in (2.2.1). Because of the assumed symmetry of the Hessian, variables $\hat{h}_{ij}$ and $\hat{h}_{ji}$ are identified. Many approximation methods determine $\hat{h}_{ij}$ and $\hat{h}_{ji}$ as if they were different, which leads to over-determined linear systems when they are identified. This is the reason why it is required that only a subsystem of (2.2.1) uniquely determine the non-zero $\hat{h}_{ij}$'s. For example, the usual method that chooses $d^i = \delta_i e^i$, $i = 1, 2, \ldots, n$, is an approximation method, where the unknown $\hat{h}_{ij}$, $i \neq j$, appears in both the j^{th} equation for d^i and the i^{th} equation for d^j. Deleting equation j for d^i when $j < i$ leads to a subsystem of (2.2.1) that can be uniquely solved for $\hat{H}(x^0)$.

For $n = 3$, this subsystem is of the form

$$\begin{pmatrix} \delta_1 & & & & & & \\ & \delta_2 & & & & & \\ & & \delta_3 & & & & \\ & & & \delta_2 & & & \\ & & & & \delta_3 & & \\ & & & & & \delta_3 & \end{pmatrix} \begin{pmatrix} \hat{h}_{11} \\ \hat{h}_{12} \\ \hat{h}_{13} \\ \hat{h}_{22} \\ \hat{h}_{23} \\ \hat{h}_{33} \end{pmatrix} = \begin{pmatrix} \Delta_1^1 \\ \Delta_2^1 \\ \Delta_3^1 \\ \Delta_2^2 \\ \Delta_3^2 \\ \Delta_3^3 \end{pmatrix}. \tag{2.2.2}$$

The efficiency of an approximation method depends on two factors. First, since evaluating the gradient is assumed to be expensive, the ideal approximation method would minimize k, the number of difference directions used to form equations (2.2.1). Finding the k difference directions is a one-time cost; since the sparsity pattern of H is independent of x, the difference directions are generated at the start of the optimization and used at every iteration thereafter.

Second, once a set of difference directions has been found, equations (2.2.1) must be solved for $\hat{H}(x^0)$. Solving a diagonal system of equations like (2.2.2) is both faster and more numerically stable than solving a general system of equations. The solution cost is incurred whenever the Hessian is to be evaluated. If an approximation method generates a set of equations (2.2.1) which are ill-conditioned, $\hat{H}$ may not be a good approximation to H, and the convergence rate of an optimization procedure may suffer. Also, if solving equations (2.2.1) is very difficult, then the approximation method may contribute an unacceptable overhead to optimization.

Thus a smaller k may lead to fewer gradient evaluations per iteration, but also to spending more time in solving equations per iteration. This trade-off has led researchers to the realization that practical approximation methods may need to restrict the form of equations (2.2.1). Powell and Toint (1979) classify approximation methods as follows. If, as in (2.2.2), the approximation method always gives rise to a diagonal subsystem, it is called a **direct method** (since the Hessian can be solved for directly). The usual unit vector method is a direct method. If the subsystem can always be permuted into triangular form, the approximation method is called a **substitution method** (since the elements of the Hessian can be solved for by simple substitution). Finally, if subsystems can arise which cannot be permuted into triangular form, the method is called an **elimination method** (since some sort of Gaussian elimination must be applied to solve for H). Examples of substitution and elimination methods are exhibited in Section 2.5.

The next section will show that it is useful to break down these classes of methods further into subclasses. Given a particular subclass S, our goal is to find a fast **optimal** method in S. That is, given a sparsity pattern H, an optimal method in S generates difference directions $d^1, d^2, \ldots, d^k$ so that (1) the restrictions of S are satisfied and (2) k is as small as possible for this H. To be practical, an optimal method must be fast in finding the d^l; although this is a one-time cost, it should not contribute too much overhead to the total optimization time. After the next section classifies the direct methods, Section 2.4 shows that in a certain technical sense, there are no fast optimal direct methods for any of the classes of direct methods.

2.3. Classifying Direct Methods

This section considers direct methods for approximating Hessians. An entry of $\hat{H}$ which is not known to be zero is called an **unknown**, and unknowns $\hat{h}_{ij}$ and $\hat{h}_{ji}$ are identified. The defining restriction of direct methods turns out to be equivalent to a relationship between the sparsity pattern of H and the zero/non-zero structure of the d^l. To see the equivalence, regard the non-zero components of each d^l as specifying a subset S_l of the column indices of H. The set S_l is called the l^{th} **group of columns** of H. When two columns belonging to S_l both have an unknown in row i, we say that there is an **overlap** in S_l in row i. For example, consider the sparsity pattern:

$$\begin{pmatrix} \times & 0 & \times \\ 0 & \times & \times \\ \times & \times & \times \end{pmatrix} \quad \leftrightarrow \quad \begin{array}{ccc} \textcircled{1} & & \textcircled{2} \\ & \diagdown\!\diagup & \\ & \textcircled{3} & \end{array}$$

The group corresponding to $d = (1, 1, 0)^T$ consists of columns $\{1, 2\}$, which overlap in row 3 but not in row 2.

The set of directions $\{ d^l \}$ chosen by a direct method gives rise to equations (2.2.1). In order for (2.2.1) to have a diagonal subsystem that can be solved for all the unknowns in H, each unknown $\hat{h}_{ij}$ (or $\hat{h}_{ji}$) must appear by itself in at least one equation, say the equation for row i in the set of equations arising from d^l. This condition implies that $j \in S_l$ and that there are no other unknowns in row i of S_l (otherwise, $\hat{h}_{ij}$ would not be the *only* unknown in the i^{th} equation of d^l). Thus, for $\hat{h}_{ij}$ to be determined directly, S_l can have no overlap in row i. The family of column subsets $\{ S_l \}$ corresponding to the directions $\{ d^l \}$ computed by a direct method must therefore satisfy the

Direct Cover Property (DCP): For each unknown $\hat{h}_{ij} = \hat{h}_{ji}$ there must be an S_l with either $j \in S_l$ and no overlap in row i, or $i \in S_l$ and no overlap in row j.

Conversely, given a sparsity pattern and a family $\{ S_l \}$ of column subsets satisfying (DCP), the set of difference directions $d^l = \sum_{j \in S_l} \delta_j e^j$ can be defined, which correspond to a direct method. Thus finding an optimal direct method is equivalent to the purely combinatorial problem of finding a minimum cardinality direct cover.

2.3.1. Approximation of Sparse Jacobians

Direct approximation methods for Hessians have evolved out of previously studied methods for approximating sparse Jacobians of functions $F: \mathrm{R}^n \to \mathrm{R}^m$ by finite-differencing. An example of the idea in this case is that if a Jacobian has the sparsity pattern

$$\begin{pmatrix} \times & 0 & 0 & \times & 0 \\ \times & 0 & 0 & 0 & \times \\ 0 & \times & 0 & 0 & \times \\ 0 & 0 & \times & \times & 0 \end{pmatrix},$$

then differencing F along $d^1 = (1,1,1,0,0)$ and $d^2 = (0,0,0,1,1)$ approximates the Jacobian in just two function evaluations rather than five, since there is no overlap among the first three or last two columns. There are also other applications of finding minimum groupings of non-overlapping columns; see, *e.g.*, Dencker, Dürre and Heuft (1981) or Dürre and Fels (1980). The possibility of such reductions in function evaluations has caused the problem of finding a minimum set of difference directions to be thoroughly studied. Several heuristics for finding "good" sets of directions have been investigated (see Curtis, Powell and Reid (1974), and Coleman and Moré (1981)). The computational complexity of finding an optimal set of difference directions in a direct Jacobian approximation method has also been investigated, resulting in the next theorem.

Theorem 2.3.1: (Coleman and Moré (1981), Theorem 3.3, and Newsam and Ramsdell (1982), Theorem 1) Finding an optimal set of difference directions for directly approximating a Jacobian is NP-Complete. □

2.3.2. Classification by Type of Overlapping

The Hessian problem is significantly more difficult than the Jacobian problem because of the symmetry of the matrix. Various heuristic approaches to finding optimal direct covers have been proposed. We review next the history of these efforts, and then propose a new way of classifying direct methods.

One obvious approach to approximating Hessians is simply to apply one of the Jacobian methods to the symmetric sparsity pattern of the Hessian. Such an approach leads to families of subsets of columns whose subsets have *no* overlap in any row. As long as every column is in some subset, such a family clearly satisfies (DCP), so that any direct Jacobian approximation method immediately becomes a direct method for Hessians. A direct cover which has no overlap in any row of any group is called a **non-overlap direct cover (NDC)**. The first NDC heuristic for Jacobians was proposed by Curtis, Powell and Reid (1974), and it was later improved by Coleman and Moré (1981).

Powell and Toint (1979) recognized that a significant decrease in gradient evaluations can be achieved by taking advantage of symmetry. For example, recall the arrowhead sparsity pattern (2.1.2). Since every column overlaps with every other, any NDC must contain at least n groups (and of course n suffice). But, as was shown in Section 2.1, if the first difference direction is $d^1 = \delta_n e^n$, then it does not matter if subsequent

columns overlap in row n since the unknowns in row n have already been determined by the first evaluation and symmetry. Exploiting this observation reduces the number of gradient evaluations for this example from n to two.

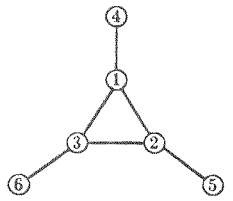
In general, if S_1 has no overlaps in any row and if $j \in S_1$, then by symmetry row j is completely determined after the first evaluation, and so overlaps in row j can be ignored in later groups. Thus we consider families of column subsets with the following property:

Sequential Overlap Property (SOP): Group S_l can have an overlap in row i only if there is a $k < l$ with $i \in S_k$.

For an unknown $\hat{h}_{ij}$, define the **minimum index group** of $\hat{h}_{ij}$ to be $p = \min\{ l \mid \text{either } i \text{ or } j \text{ belongs to group } l \}$. When (SOP) holds, $\hat{h}_{ij}$ must be the only unknown in its row in group p, so that (DCP) is satisfied. Direct covers with (SOP) are called **sequential overlap direct covers (SeqDC)**. Note that any NDC algorithm that generates its groups sequentially can easily be converted into a SeqDC algorithm by deleting the columns and corresponding rows of group S_l before finding group S_{l+1}.

Powell and Toint (1979) showed that there are sparsity patterns for which an optimal SeqDC is not an optimal direct cover. Their example is:

$$\begin{pmatrix} \times & \times & \times & \times & 0 & 0 \\ \times & \times & \times & 0 & \times & 0 \\ \times & \times & \times & 0 & 0 & \times \\ \times & 0 & 0 & \times & 0 & 0 \\ 0 & \times & 0 & 0 & \times & 0 \\ 0 & 0 & \times & 0 & 0 & \times \end{pmatrix} \leftrightarrow$$



(2.3.1)

It is easy to see that any SeqDC for (2.3.1) requires at least four groups, but that $\{\{1,5\},\{2,6\},\{3,4\}\}$ is a direct cover of size only three. Thapa (1980), Section 5.1, proposed a method that tries to take advantage of such situations, and that produces a direct cover which may not satisfy (SOP). Direct covers that do not necessarily satisfy any additional restrictions are called **simultaneous overlap direct covers** (SimDC); *any* direct cover is a SimDC.

Note that NDC $\subset$ SeqDC $\subset$ SimDC and that these inclusions are strict, as shown by (2.1.2) and (2.3.1).

2.3.3. Classification by Partitioning

All of the heuristic methods mentioned above produce **partitioned** direct covers, that is, every column belongs to exactly one group. Even when there are no zero columns, not every column need belong to some group in a valid direct cover. Consider

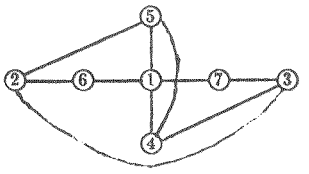
$$\begin{pmatrix} 0 & \times \\ \times & \times \end{pmatrix};$$

$\{\{2\}\}$ is a direct cover (in fact, an NDC) of minimum size, yet column one belongs to no group. But when $\hat{h}_{ii}$ is an unknown, column i must belong to some group in order for $\hat{h}_{ii}$ to be determined. When H arises from unconstrained optimization, $\hat{h}_{ii}$ is usually non-zero for all i, which implies that all columns must be in some group. Thus, unless otherwise stated, henceforth it is assumed that $\hat{h}_{ii}$ is an unknown for all i, so that only direct covers containing every column need to be considered.

Now consider a SeqDC in which columns may appear in more than one group. If every occurrence of every column in a group other than its smallest index group is deleted, (SOP) holds since each unknown is determined by its minimum index group. That is, since each unknown is determined by a column in its minimum index group, any later occurrences of that column are superfluous, and there can be no occurrences of that column in an earlier group by the definition of minimum index group. Thus, for SeqDCs (and also for NDCs since NDC $\subset$ SeqDC), it suffices to consider only partitioned direct covers.

Unfortunately, the same is not true for SimDCs. Consider the sparsity pattern:

$$\begin{pmatrix} \times & 0 & 0 & \times & \times & \times & \times \\ 0 & \times & \times & 0 & \times & \times & 0 \\ 0 & \times & \times & \times & 0 & 0 & \times \\ \times & 0 & \times & \times & \times & 0 & 0 \\ \times & \times & 0 & \times & \times & 0 & 0 \\ \times & \times & 0 & 0 & 0 & \times & 0 \\ \times & 0 & \times & 0 & 0 & 0 & \times \end{pmatrix} \leftrightarrow$$



Laborious calculations verify that any **partitioned simultaneous direct cover (PSimDC)** must use at least five groups, whereas $\{\{1,2\},\{1,3\},\{4,6\},\{5,7\}\}$ is a **general simultaneous direct cover (GSimDC)** which uses only four groups. This is the smallest possible such example in terms of number of columns. Eisenstat (referenced in Coleman and Moré (1982), equation (2.1)) has discovered an infinite class of such examples.

These two classifications give four distinct subclasses of direct covers: NDC, SeqDC, PSimDC and GSimDC. The next section shows that finding an optimal member each of the four classes is NP-Complete.

2.4. The Complexity of Direct Methods

The main purpose of this section is to prove the following four theorems:

Theorem 2.4.1: Finding an optimal NDC is NP-Complete.

Theorem 2.4.2: Finding an optimal SeqDC is NP-Complete.

Theorem 2.4.3: Finding an optimal PSimDC is NP-Complete.

Theorem 2.4.4: Finding an optimal GSimDC is NP-Complete.

Recall from Section 1.2 that to prove that a problem X is NP-Complete it is necessary to reduce a known NP-Complete problem to problem X. We shall use three known NP-Complete problems. The first is the direct Jacobian approximation problem already discussed in Section 2.3. The second is the

3-Satisfiability Problem (3SAT): Let $u_1, u_2, \ldots, u_n$ be a set of **atoms**, with the corresponding set of **literals** $L = \{u_1, \bar{u}_1, u_2, \bar{u}_2, \ldots, u_n, \bar{u}_n\}$. Let $C = \{C_1, C_2, \ldots, C_m\}$ be a set of **3-clauses** drawn from L, that is, each $C_s \subset L$, and $|C_s| = 3$. Is there a **truth assignment** $\tau : \{u_1, \ldots, u_n\} \rightarrow \{true, false\}$ such that each C_s contains at least one u_i with $\tau(u_i) = true$ or at least one $\bar{u}_i$ with $\tau(u_i) = false$?

The set of clauses is an abstraction of a logical formula; imagine the clauses as parenthesized subformulae whose literals are connected by 'or', with all the clauses connected by 'and'. Then a satisfying truth assignment makes the whole formula true.

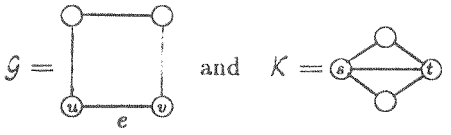
The third problem is the

3-Color Graph Coloring Problem: Given a graph $\mathcal{G} = (V, E)$, does there exist a function $f: V \rightarrow \{1,2,3\}$ such that $f(v) \neq f(u)$ whenever $\{v,u\} \in E$?

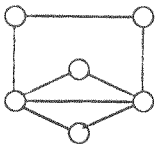
This problem remains NP-Complete even when $\mathcal{G}$ is restricted to be planar (see Garey and Johnson (1979), Theorem 4.2). Note that 3GCP is a restricted form of the classic problem of finding the chromatic number of a graph. The NP-Completeness of the Jacobian direct approximation problem was stated in Theorem 2.3.1. The NP-Completeness proofs of 3SAT and 3GCP are referenced in Garey and Johnson (1979), Problems [LO2] and [GT4] respectively.

A graph operation that is needed in two of the proofs is the notion of **edge replacement**. Given a graph $\mathcal{G}$ and edge $e = \{u, v\}$ in $\mathcal{G}$, and graph K with two distinguished vertices s and t called **terminals**, the result of replacing edge e of $\mathcal{G}$ by K is obtained by removing e from $\mathcal{G}$ and identifying u with s and v

with t. For example, if



then replacing e with K yields the graph:



All the proofs depend on the equivalence between Hessian sparsity patterns and undirected graphs mentioned in Section 2.1. Throughout this section, $\mathcal{G}$ denotes the graph associated with H.

If a polynomial algorithm exists for finding an optimal direct cover of any type, it must in particular work on sparsity patterns whose diagonal entries are all unknowns. Thus it can be assumed without loss of generality that every column index belongs to at least one group. A direct cover then associates with each vertex of $\mathcal{G}$ the index of the group (indices of groups in the case of GSumDCs) to which its column belongs. Such an association of integers to vertices is a graph coloring, whose type depends on the nature of the associated direct cover. In each of the four cases, we shall establish what sort of coloring is involved and show that the problem is NP-Complete Note that in every case two adjacent vertices i and j cannot be the same color because then neither $\hat{h}_{ii}$ nor $\hat{h}_{jj}$ could be determined directly due to overlap with $\hat{h}_{ij}$. Thus the generalized graph coloring must be a usual graph coloring as well.

2.4.1. The Complexity of Finding Optimal NDCs

We shall give two proofs of Theorem 2.4.1, the first very general and complicated, the second quite specific and short.

In an NDC, two columns in the same group, *i.e.* two vertices of the same color, cannot overlap. Column i overlaps column j if $\hat{h}_{ki}$ and $\hat{h}_{kj}$ are both unknowns, *i.e.*, if vertices i and j are both adjacent to vertex k. Thus in the coloring of $\mathcal{G}$, no two vertices of the same color can have a common neighbor. Conversely, given such a coloring of $\mathcal{G}$, clearly no two columns in the associated groups can overlap.

If distance from vertex i to vertex j in the graph is measured by "minimum number of edges in any path between i and j", then any two vertices of the same color must be more than two units apart. In the usual Graph Coloring Problem, any two vertices of the same color must be more than one unit apart. Then a common generalization is a **proper distance-k coloring** of a graph $\mathcal{G}$, which is a partition of the vertices of $\mathcal{G}$ into classes (colors) so that every pair of vertices of the same color is more than k units apart. The associated optimization problem is the

Distance-k Graph Coloring Problem (DkGCP): Given a graph $\mathcal{G}$, find a proper distance-k coloring of $\mathcal{G}$ in the minimum possible number of colors.

The usual Graph Coloring Problem (GCP) is D1GCP, and the optimal NDC problem is equivalent to D2GCP. We shall use this equivalence to show that the optimal NDC problem is NP-Complete by showing that D2GCP is NP-Complete; in fact, we shall show the stronger result that DkGCP is NP-Complete for any fixed $k \geq 2$.

To show that DkGCP is NP-Complete, 3SAT will be encoded into it. To facilitate the encoding, DkGCP must be recast as a decision problem. As is standard with optimization problems, DkGCP is re-phrased to "Is there a distance-k coloring that uses p or fewer colors?" In a slight abuse of notation, let DkGCP refer to both the optimization problem and the related decision problem. Our encoding is a generalization of the one found in Karp's original proof (1972) of the NP-Completeness of GCP. The first proof of Theorem 2.4.1 requires the exclusion of the case in which a clause contains both an atom and its negation. But such clauses are always trivially satisfied, and so henceforth "3SAT" will mean "3-Satisfiability without trivial clauses" Also, it is assumed (without loss of generality) that $n \geq 4$. Then for any clause C_j, there is an atom u_i with $u_i \notin C_j$, and $\overline{u}_i \notin C_j$.

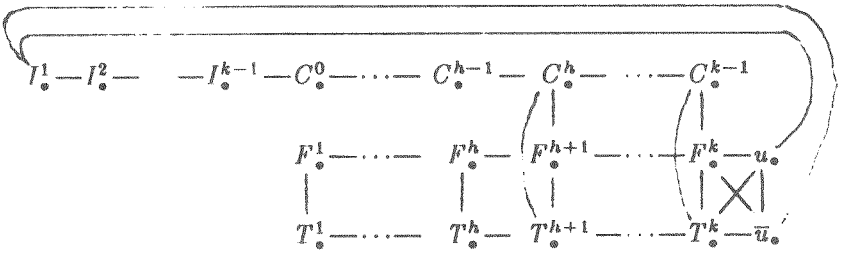
Given a 3SAT problem P, we construct from it a decision problem on a graph $\mathcal{G}_k(P)$. If P has atoms $u_1, u_2, \ldots, u_n$ and clauses $C_1, C_2, \ldots, C_m$, let $h = \lceil k/2 \rceil$ and $p = 2nk + m(k-1)$. Let V and E denote the vertices and edges of $\mathcal{G}_k(P)$, and define them by:

$$V = \left\{ \begin{array}{l} u_i, \overline{u}_i \\ F_i^r, T_i^r, \quad r = 1, \ldots, k \end{array} \right\} \; i = 1, \ldots, n \quad \text{literal vertices, false vertices, true vertices}$$

$$\left\{ \begin{array}{l} C_s^r, \quad r = 0, \ldots, k-1 \\ I_s^r, \quad r = 1, \ldots, k-1 \end{array} \right\} \; s = 1, \ldots, m \quad \text{clause vertices, intermediate vertices}$$

$$E = \left\{ \begin{array}{ll} \{u_i, \overline{u}_i\} & \text{all } i \qquad u_i, \overline{u}_i \text{ different colors} \\ \left. \begin{array}{l} \{F_i^r, F_i^{r+1}\} \\ \{T_i^r, T_i^{r+1}\} \\ \{F_i^r, T_j^r\} \\ \{F_i^r, F_j^r\} \\ \{T_i^r, T_j^r\} \end{array} \right\} & \text{all } r, \text{ all } i \neq j \qquad \text{all } F\text{'s, } T\text{'s different colors} \\ \left. \begin{array}{ll} \{I_s^r, I_s^{r+1}\} & \text{all } r \\ \{I_s^{k-1}, C_s^0\} & \\ \{I_s^1, u_i\} & \text{if } u_i \in C_s \\ \{I_s^1, \overline{u}_i\} & \text{if } \overline{u}_i \in C_s \end{array} \right\} & \text{all } s \qquad C_s^0 \text{ different color than its literals} \\ \left. \begin{array}{l} \{u_i, F_j^k\} \\ \{u_i, T_j^k\} \\ \{\overline{u}_i, F_j^k\} \\ \{\overline{u}_i, T_j^k\} \end{array} \right\} & \text{all } i \neq j \qquad u_i, \overline{u}_i \text{ can only be } F_i^1 \text{ or } T_i^1 \\ \left. \begin{array}{ll} \{C_s^r, C_t^r\} & r \geq h \\ \{C_s^r, F_i^{r+1}\} & r \geq h \\ \{C_s^r, T_i^{r+1}\} & r \geq h \\ \{C_s^r, C_s^{r+1}\} & r \geq 0 \end{array} \right\} & \text{all } s \neq t, \text{ all } i \qquad C_i^r, r > 0, \text{ different from each other and } F\text{'s and } T\text{'s} \\ \left. \begin{array}{ll} \{C_s^{h-1}, F_l^h\} & u_l, \overline{u}_l \notin C_s \\ \{C_s^{h-1}, T_i^h\} & \text{all } i \end{array} \right\} & \text{all } s, k \text{ odd} \\ \left. \begin{array}{ll} \{C_s^h, F_l^h\} & u_l, \overline{u}_l \notin C_s \\ \{C_s^h, T_i^h\} & \text{all } i \end{array} \right\} & \text{all } s, k \text{ even} \end{array} \right\} \qquad C_s^0 \text{ can only be } F^1 \text{ colors of its literals}$$

Note that $\mathcal{G}_k(P)$ is considered only for $k \geq 2$, implying that $h + 1 \leq k$, so that $h + 1$ makes sense as a superscript for the F's and the T's. The global structure of $\mathcal{G}_k(P)$ looks like



Three propositions about the structure of a proper distance-k coloring of $\mathcal{G}_k(P)$ are needed for the proof.

Proposition 2.4.5: The vertices F_i^r, T_i^r, and C_s^{r-1}, $i = 1, \ldots, n$, $s = 1, \ldots, m$, $r = 1, \ldots, k$ must all have different colors, thus using up all p colors.

Proof: Consider the length k paths

$$\left. \begin{array}{l} F_i^1 \\ T_i^1 \end{array} \right\} - \left\{ \begin{array}{l} F_j^1 - F_j^2 - \cdots - F_j^k \\ T_j^1 - T_j^2 - \cdots - T_j^k \end{array} \right.$$

which demonstrate that all F's and T's must be different colors. Now consider the length $k-1$ paths

$$C_s^1 — C_s^2 \ — \cdots — C_s^h — \begin{cases} F_i^{h+1} — F_i^{h+2} \ — \cdots — F_i^k \\ T_i^{h+1} — T_i^{h+2} \ — \cdots — T_i^k \end{cases} \tag{2.4.1}$$

which show that no C_s^q, $0 < q \leq h$ can be any F_i^r or T_i^r color, $r > h$. The length at most $k-1$ paths

$$C_s^{k-1} — C_s^{k-2} \ — \cdots — C_s^h — \begin{cases} F_i^{h+1} — F_i^{h+2} \ — \cdots — F_i^k \\ T_i^{h+1} — T_i^{h+2} \ — \cdots — T_i^k \end{cases}$$

show that no C_s^q, $h \leq q < k$ can be any F_i^r or T_i^r color, $r > h$. Let l be an index such that $u_l \notin C_s$ and $\overline{u}_l \notin C_s$, and consider the length $k-1$ paths

$$C_s^1 — \cdots — C_s^{h-1} — \begin{cases} T_i^h — T_i^{h-1} \ — \cdots — T_i^1 & k \text{ odd} \\ C_s^h — T_i^h \ — \cdots — T_i^1 & k \text{ even} \end{cases} \tag{2.4.2}$$

which show that no C_s^q, $0 < q < h$ can be any T_i^r color, $r \leq h$. The length k paths

$$C_s^1 — C_s^2 \ — \cdots — C_s^{h-1} — \begin{cases} F_l^h — F_i^h \ — F_i^{h-1} — \cdots — F_i^1 & k \text{ odd} \\ C_s^h — F_l^h \ — F_i^h \ — \cdots — F_i^1 & k \text{ even} \end{cases} \tag{2.4.3}$$

show that no C_s^q, $0 < q < h$ can be any F_i^r color, $r \leq h$. The length k paths

$$C_s^{k-1} — C_s^{k-2} \ — \cdots — C_s^h — \begin{cases} F_i^{h+1} — F_i^h \ — \cdots — F_i^1 \\ T_i^{h+1} — T_i^h \ — \ — T_i^1 \end{cases}$$

show that no C_s^q, $h \leq q < k$, can be any F_i^r or T_i^r color, $r \leq h$. The length $k-1$ paths

$$C_s^1 — C_s^2 \ — \cdots — C_s^{h-1} — \begin{cases} T_i^h — C_t^{h-1} \ — C_t^{h-2} — \cdots — C_t^1 & k \text{ odd} \\ C_s^h — C_t^h \ — C_t^{h-1} — \cdots — C_t^1 & k \text{ even} \end{cases} \tag{2.4.4}$$

show that no C_s^q can be the same color as any C_t^r, $0 < q$, $r < h$. Finally, the length k 1 path

$$C_s^1 — C_s^2 \ — \cdots — C_s^h — C_t^h — C_t^{h+1} \ — \cdots — C_t^{k-1} \tag{2.4.5}$$

shows that no C_s^q can be the same color as any C_t^r, $h \leq q$, $r < k$. □

Since F_i^r, T_i^r and C_s^q, $q > 0$, use up all the colors, the colors are subsequently referred to by these vertex names.

Proposition 2.4.6: Vertices u_i and $\overline{u}_i$ must be colored F_i^1 and T_i^1 in some order, $i = 1, \ldots, n$.

Proof: Let $j \neq i$ and consider the length k paths

$$\left.\begin{matrix} u_i \\ \overline{u}_i \end{matrix}\right\} — \begin{cases} F_j^k — \begin{cases} F_i^k \ — F_i^{k-1} \ — \cdots — F_i^2 \\ F_j^{k-1} — F_j^{k-2} \ — \cdots — F_j^1 \end{cases} \\ T_j^k — \begin{cases} T_i^k \ — T_i^{k-1} \ — \cdots — T_i^2 \\ T_j^{k-1} — T_j^{k-2} \ — \cdots — T_j^1 \\ C_s^{k-1} — C_s^{k-2} \ — \cdots — C_s^1 \end{cases} \end{cases} \tag{2.4.6}$$

which show that u_i and $\overline{u}_i$ cannot be any color other than F_i^1 and T_i^1. Also, u_i and $\overline{u}_i$ certainly cannot be the same color. □

Thus a proper distance-k coloring of $\mathcal{G}_k(P)$ induces a truth assignment on the literals.

Proposition 2.4.7: If the literals in clause C_s have indices a, b, and c, then C_s^0 must be colored F_a^1, F_b^1, or F_c^1, $s = 1, \ldots, m$.

Proof: Note that C_s^0 can be added to the beginning of the paths in (2.4.1), (2.4.2), (2.4.3), (2.4.4) and (2.4.5), thus excluding all colors except F_i^1 from C_s^0. If l is an index such that $u_l \notin C_s$, $\overline{u}_l \notin C_s$, then the edge $F_l^h - F_i^h$ can be dropped from (2.4.3) and C_s^0 can be added to the beginning to show that C_s^0 cannot be any F_l^1 color either. □

Now the NP-Completeness theorem can be stated and proved, which will then immediately imply that finding an optimal NDC is NP-Complete. This theorem is a symmetric version of a result in Section 3 of Coleman and Moré (1981).

Theorem 2.4.8: For fixed $k \geq 2$, DkGCP is NP-Complete.

Proof: Since the size of $\mathcal{G}_k(P)$ is a polynomial in m and n, it is clear that the above reduction of 3SAT to DkGCP can be carried out in polynomial time. It must be shown that there is a satisfying truth assignment for the 3SAT problem P if and only if the graph $\mathcal{G}_k(P)$ has a proper distance-k coloring in p or fewer colors.

First suppose that $\mathcal{G}_k(P)$ is properly distance-k colored. If l_a, l_b, and l_c are the literals contained in C_s, then the length k path

$$C_s^0 - I_s^{k-1} - I_s^{k-2} - \cdots - I_s^1 - l_i$$

shows that C_s^0 cannot be the same color as any of l_a, l_b, or l_c. But C_s^0 must be colored F_a^1, F_b^1, or F_c^1 by Proposition 2.4.7. By Proposition 2.4.6, each l_i is colored either F_i^1 or T_i^1, so that each clause must contain at least one true literal under the truth assignment induced by the proper coloring, *i.e.*, the clauses are satisfiable.

Now it suffices to show that $\mathcal{G}_k(P)$ can always be colored in p or fewer colors if P is satisfiable. Let τ be a satisfying truth assignment for $C_1, C_2, \ldots, C_m$. First color the F_i^r's, T_i^r's and C_s^r's, $r > 0$, as decreed by Proposition 2.4.5. Color u_i with T_i^1 if $\tau(u_i) = \mathit{true}$, color u_i with F_i^1 otherwise; color $\overline{u}_i$ with the complementary color. Each C_s has at least one true literal, say l_a. Color C_s^0 with color F_a^1. Finally, color I_s^r with C_{s+1}^r, $r = 1, \ldots, k-1$, where the subscript on C_s^r is interpreted modulo m.

We now show that this coloring is proper. The colors F_i^r, T_i^r, $1 < r \leq k$, each appear on only one vertex and so are proper. Color C_{s+1}^r appears on exactly two vertices, itself and I_s^r. A shortest possible path between these vertices in $\mathcal{G}_k(P)$ is

$$I_s^r - I_s^{r-1} - \cdots - I_s^1 - \left\{ \begin{matrix} u_i \\ \overline{u}_i \end{matrix} \right\} - T_j^k - C_{s+1}^{k-1} - C_{s+1}^{k-2} - \cdots - C_{s+1}^r$$

and is of length $k+1$. It is a shortest path because at least k edges must be used to get from layer I_*^r to layer C_*^r, and one extra edge must be used to get from an F or a T to a C. Also, any alternative path between these vertices that goes through a C_*^0 has at least $k+2h$ edges because of the difference in subscripts, and because the C_s^r's do not interconnect for $r < h$; thus color C_s^r is proper. Color T_i^1 also appears on exactly two vertices, itself and one of u_i or $\overline{u}_i$. A shortest possible path between these vertices is the third one in (2.4.6) with T_i^1 added at the end. For $j \neq i$, at least k edges must be used to get from the u_* layer to the T_*^1 layer, and an extra edge is necessary to go from an i vertex to a j vertex. Any other path between these vertices through the I's uses at least $k+2h-1$ edges, and so T_i^1 is proper. Finally, F_i^1 can appear in three places: on itself, on u_i or $\overline{u}_i$, and on any number of C_s^0's whose clauses contain either u_i or $\overline{u}_i$. As with u_i or $\overline{u}_i$ and T_i^1 above, u_i or $\overline{u}_i$ and F_i^1 do not cause a conflict. Some shortest possible paths between F_i^1 and any C_s^0 are those in (2.4.3) with C_s^0 added to the beginning. Again, at least k edges are necessary to go from the C_*^0 layer to the F_*^1 layer, and an extra edge is necessary to go from an l vertex to an i vertex. Any other path between these vertices through the I's uses at least $2k$ edges, so that no C_s^0, F_i^1 pair causes a conflict. Between a u_i or $\overline{u}_i$ and a C_s^0, some shortest possible paths are

$$\left\{ \begin{matrix} u_i \\ \overline{u}_i \end{matrix} \right\} - \left\{ \begin{matrix} I_s^1 - I_s^2 - \cdots - I_s^{k-1} - C_s^0 \\ T_j^k - C_s^{k-1} - \cdots - C_s^1 - C_s^0 \end{matrix} \right.$$

of lengths k and $k+1$ respectively. The first cannot exist because of the truth assignment and because there are no trivial clauses. Once again, the second must use k edges going from layer u_* to layer C_*^0, and an extra

edge going from an F or a T to a C, so that no u_i or $\overline{u}_i$, C_s^0 pair conflicts. Finally, a shortest possible path between C_s^0 and C_t^0 is (2.4.4) with C_s^0 added to the beginning and C_t^0 added to the end, of length $k+1$. Any other path between these vertices through the I's uses at least $2k$ edges, so that no F_i^1 color conflicts. Thus the coloring is proper, and the theorem is proved. □

Second Proof of 2.4.1: This proof is due to Hoffman (1982). The direct Jacobian approximation problem is reduced to finding an optimal NDC.

Given an $m \times n$ Jacobian sparsity pattern A, consider the symmetric $(m+n) \times (m+n)$ matrix

$$H = \begin{pmatrix} J & A \\ A^T & I \end{pmatrix},$$

where J is the $n \times n$ matrix of all ones, and I is the $m \times m$ identity. Suppose that there is a polynomial algorithm for finding optimal NDCs, and apply it to H. Since each of the first m columns of H overlaps with all other columns, each of the first m columns must appear by itself; but finding an optimal NDC for H then essentially reduces to finding a minimum partition of the last n columns of H into non-overlapping groups. However, such a minimum partition solves the direct Jacobian approximation problem, which is NP-Complete. Thus finding an optimal NDC must be NP-Complete as well. □

2.4.2. The Complexity of Finding Optimal SeqDCs

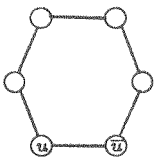
We now consider the sort of coloring of $\mathcal{G}$ induced by a SeqDC. The first color must be non-overlapping; hence, as in the NDC case, no two color 1 vertices can have a common neighbor. Now, since overlap in the group 1 rows no longer matters, the row and column indices in group 1 can be deleted from H, and the group 2 columns can have no overlap in the *reduced* H. In graph terms, the reduction of the matrix corresponds to deleting the color 1 vertices (and their incident edges) from $\mathcal{G}$ and requiring that the color 2 vertices have no common neighbors in the reduced $\mathcal{G}$. The color 2 vertices are then deleted from the graph, and so on. Thus a SeqDC with k groups is equivalent to a

Sequential k-Coloring: A sequential k-coloring of a graph $\mathcal{G}$ is a function $f: V \to \{1, 2, \ldots, k\}$ such that no two vertices u and v with $f(u) = f(v) = l$ have a common neighbor in the graph obtained by deleting all vertices w with $f(w) < l$ from $\mathcal{G}$.

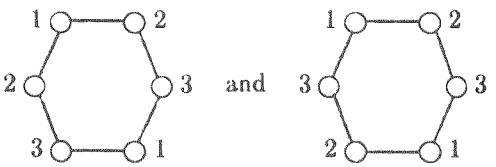
We shall show that it is NP-Complete to decide whether a graph $\mathcal{G}$ has a sequential 3-coloring.

Proof of Theorem 2.4.2: This proof is due to Stockmeyer (1982). We shall reduce 3SAT to the problem of deciding whether $\mathcal{G}$ has a sequential 3-coloring. By the equivalence between SeqDCs and sequential graph coloring, the reduction will show that finding an optimal SeqDC is also NP-Complete.

Given an instance P of 3SAT, a graph $\mathcal{G}$ will be constructed such that P has a satisfying truth assignment if and only if there is a sequential 3-coloring of $\mathcal{G}$. For each atom u of P, make a 6-cycle in $\mathcal{G}$ with two adjacent vertices labelled u and $\overline{u}$, as follows:

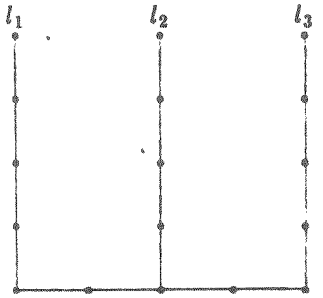


By exhaustive enumeration it can be verified that the only two possible ways to 3-color the above graph properly and sequentially are:



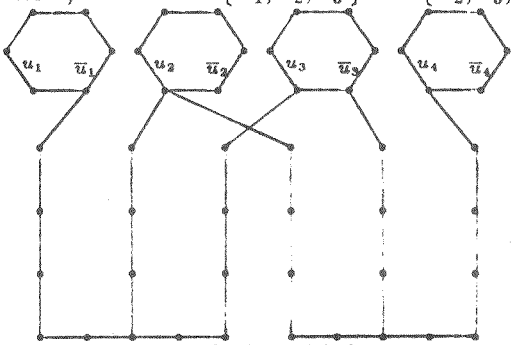
(2.4.7)

Connect these 6 cycles according to the clauses of P as follows. For each clause $C_j = \{l_1, l_2, l_3\}$, add the nodes and edges

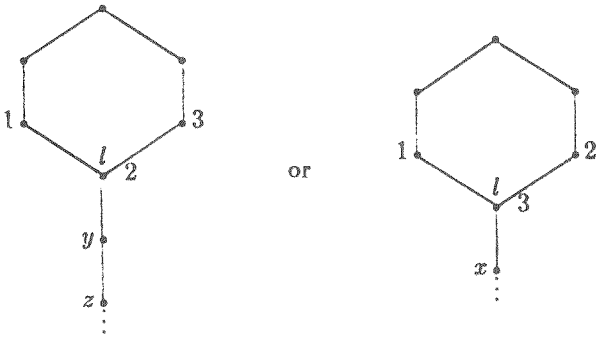


to $\mathcal{G}$, where l_1, l_2 and l_3 are the literal vertices on the 6-cycles.

For example, if P has four atoms, and clauses $\{\overline{u}_1, u_2, u_3\}$ and $\{u_2, \overline{u}_3, u_4\}$, then the constructed $\mathcal{G}$ is:



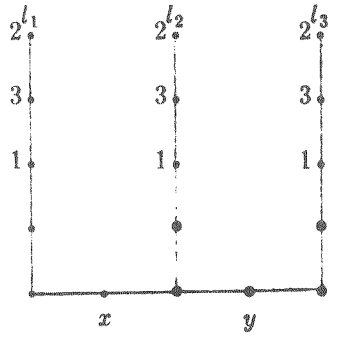
First note that by (2.4.7), the neighborhood of a literal l that is in some clause and is not colored 1 must look like:



(2.4.8)

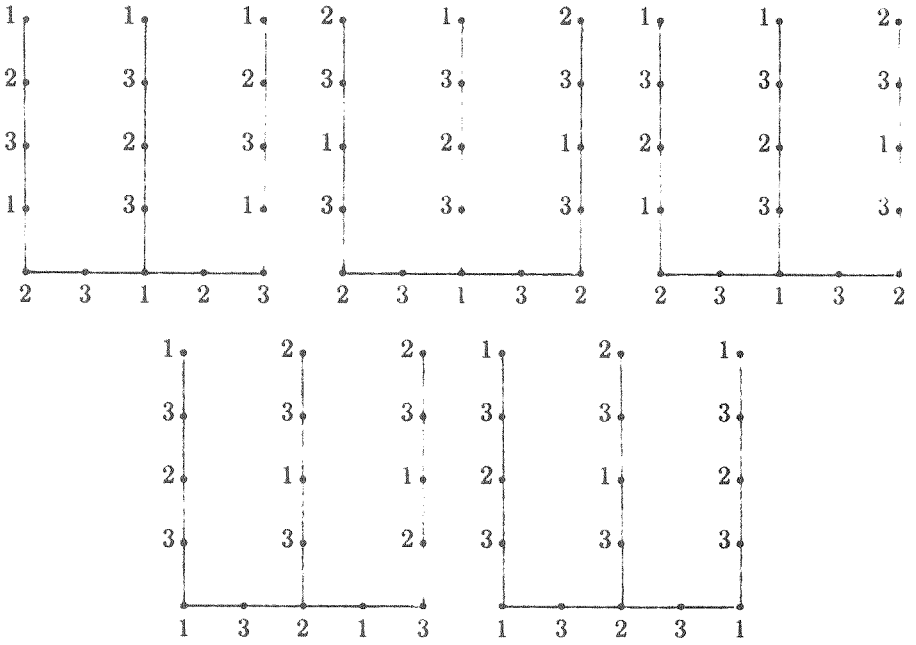
In the second case, vertex x cannot be colored any of 1, 2 or 3 in a proper sequential 3-coloring, and so every literal that appears in some clause must be colored 1 or 2. In the first case of (2.4.8), vertex y must be colored 3, and hence vertex z must be colored 1.

Suppose that $\mathcal{G}$ has been properly sequentially 3-colored and that all three literals in a clause are colored 2. Then by the above remarks about (2.4.8), the clause vertices must be partly colored as follows:

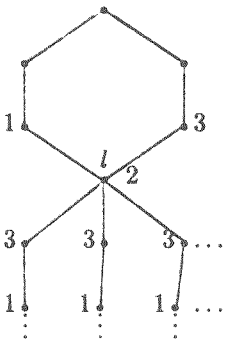


The only remaining vertices that could be colored 1 are x and y, and at most one of them can be colored 1. But then deleting the color 1 vertices leaves a path of length at least five (the darker vertices if x, say, is colored 1) which cannot be sequentially 2-colored. Thus every clause must have at least one literal colored 1. If a truth assignment is associated with the coloring by setting atom u_i *true* if vertex u_i is colored 1, and *false* otherwise, then the existence of the sequential 3-coloring of $\mathcal{G}$ implies that P has a satisfying truth assignment.

Conversely, suppose that P has a satisfying truth assignment τ. Color literal vertex l of $\mathcal{G}$ with 1 if $\tau(l) = true$, and with 2 otherwise. Arbitrarily extend the coloring as in (2.4.7) to the rest of the atomic 6-cycles. Because of the asymmetry of the clause subgraphs, there are five cases to consider in showing how to color the clause subgraphs, depending on which subset of the literals is true. The five cases and their colorings are:



The only non-trivial case to check in verifying that this coloring is a proper and sequential 3-coloring of $\mathcal{G}$ is in the neighborhood of a false literal l that is in many clauses. By (2.4.7) and (2.4.8) it must look like:



The correctness of the coloring is easily seen. Thus the existence of a satisfying truth assignment for P implies the existence of a proper sequential 3-coloring for $\mathcal{G}$. □

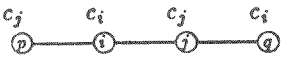
2.4.3. The Complexity of Finding Optimal PSimDCs

We now consider the sort of coloring a PSimDC give rise to. By (DCP), for each unknown $\hat{h}_{ij}$ of H either the group containing i cannot have overlap in row j, or the group containing j cannot have overlap in

row i. Equivalently, a family of subsets of columns is *not* a direct cover if there is an unknown $\hat{h}_{ij}$ such that there is overlap in row j of the group containing i, and there is overlap in row i of the group containing j. In graph terms, the coloring is improper if there is an edge $e = \{i, j\}$ in $\mathcal{G}$ where i is colored c_i, j is colored c_j, i is adjacent to a vertex $p \neq j$ also colored c_j (corresponding to overlap in row i of group c_j containing j), and j is adjacent to a vertex $q \neq i$ also colored c_i.

This "excluded colored subgraph" condition is clearly equivalent to (DCP), and so the next definition is equivalent to a PSimDC.

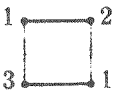
Direct k-Coloring: A direct k-coloring of a graph $\mathcal{G}$ is a function $f: V \to \{1, 2, \ldots, k\}$ such that f is a coloring in the usual sense and there is no subgraph of $\mathcal{G}$ colored like:



where $i \neq q$ and $j \neq p$.

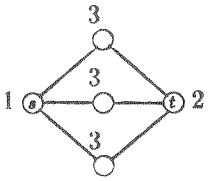
Proof of Theorem 2.4.3: We shall reduce 3GCP to the problem of deciding whether $\mathcal{G}$ has a proper direct 3-coloring. Since direct k-coloring is equivalent to finding an optimal PSimDC, the reduction will imply that finding an optimal PSimDC is NP-Complete.

Given a graph K for 3GCP, a graph $\mathcal{G}$ will be constructed such that K has a 3-coloring if and only if $\mathcal{G}$ has a direct 3-coloring. First note that the 4-cycle has essentially only one proper direct 3-coloring, up to permutation of colors:



(2.4.9)

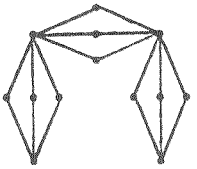
By (2.4.9), again up to a permutation of colors, there is essentially only one way to color the graph:

$L =$ 

(2.4.10)

Thus graph L forces its terminal s and t to be different colors.

To construct $\mathcal{G}$ from K, replace *every* edge of K with L. Thus if

$K =$  then $\mathcal{G} =$ 

Suppose that $\mathcal{G}$ has a proper direct 3-coloring f. By (2.4.10), if each vertex of K is colored with the color received by its identified terminal under f, the resulting coloring must be a proper 3-coloring of K.

Conversely, suppose that there is a proper 3-coloring f of K. Then color each terminal of $\mathcal{G}$ with the color of its identified vertex in K, and color each non-terminal with the complementary color of the colors of the two terminals to which it is adjacent. This coloring is clearly a proper 3-coloring of $\mathcal{G}$. Any path of four vertices in $\mathcal{G}$ must contain two terminals separated by a non-terminal. But these three vertices use all three colors, and hence the excluded colored subgraph of $\mathcal{G}$ cannot exist. Thus $\mathcal{G}$ has a proper direct 3-coloring. □

A direct coloring is called a "symmetric coloring" in Coleman and Moré (1982), and they call a PSimDC a "symmetrically consistent partition." They use these concepts to give a quite different proof of Theorem 2.4.3 (see their Theorem 3.3 and the remarks following it).

2.4.4. The Complexity of Finding Optimal GSimDCs

The fact that a column can belong to more than one group, and hence a vertex of $\mathcal{G}$ can receive more than one color, makes the proof of Theorem 2.4.4 more difficult than the proof of Theorem 2.4.3. The same analysis of the implications of (DCP) for colorings as in the PSimDC case still holds here except that the concept "vertex i is colored c_i" is replaced by "color c_i is one of vertex i's colors." With this insight it can be shown that the following formal definition is equivalent to a GSimDC with k groups. Let $S = \{1, 2, \ldots, k\}$.

Direct k-Multicoloring: A direct multicoloring of a graph $\mathcal{G}$ is a function $f: V \to 2^S \setminus \emptyset$ satisfying (1) for each edge $e = \{i, j\}$ of $\mathcal{G}$, $f(i) \cap f(j) = \emptyset$ (this is analogous to f being a coloring in the usual sense) and (2) there is no subgraph of $\mathcal{G}$ like

with $i \neq q$, $j \neq p$, $f(i) \cap f(q) \neq \emptyset$ and $f(j) \cap f(p) \neq \emptyset$ (this is analogous to the excluded colored subgraph condition for PSimDCs).

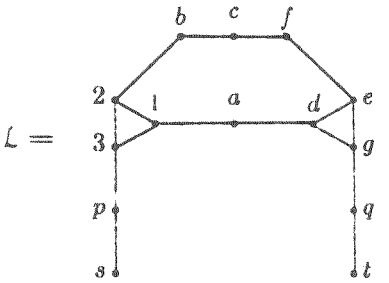
Proof of Theorem 2.4.4: By reduction from 3GCP. Because of the equivalence of direct k-multicoloring with finding optimal GSimDCs discussed above, it suffices to reduce 3GCP to the problem of deciding whether a graph has a proper direct 3-multicoloring.

Given a graph K for 3GCP, a graph $\mathcal{G}$ will be constructed such that K has a proper 3-coloring if and only if $\mathcal{G}$ has a proper direct 3-multicoloring. First, note that by exhaustive enumeration, up to a permutation of colors the only way to 3-multicolor the displayed graph properly and directly is as shown:

(2.4.11)

In particular, all vertices of subsequent graphs constructed with (2.4.11) can receive only one color.

Given the essentially unique coloring of (2.4.11), consider how to extend the indicated partial coloring on the following graph:



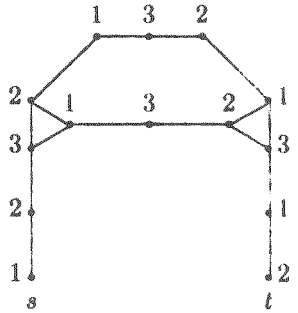
Now using "$x = i$" as shorthand for "vertex x is colored i" it can be seen that

$$a = 2 \Rightarrow \left\{ \begin{array}{c} b = 3 \Rightarrow c = 1 \\ d = 3 \end{array} \right\} \Rightarrow e = 2 \Rightarrow f = 3,$$

which implies that path $adef$ is colored 2323, an improper 3-multicoloring, and so a must be colored 3 and d must be colored 2. Now

$$b = 3 \Rightarrow c = 1 \Rightarrow e = 3 \Rightarrow f = 2,$$

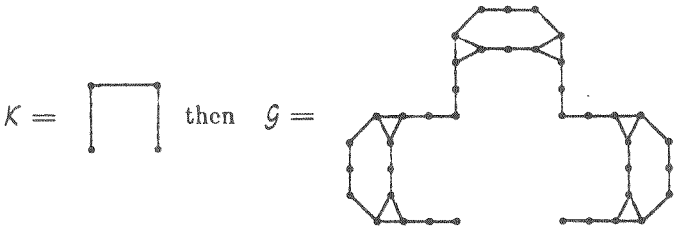
which implies that path $adcf$ is colored 3232, so that b must be colored 1. Thus the only way (up to a permutation) to 3-multicolor L properly is:



(2.4.12)

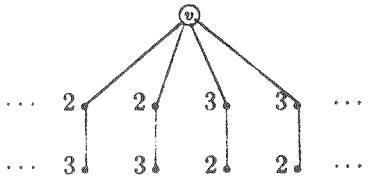
Note that as in (2.4.10), L forces its terminals s and t to be different colors.

To derive $\mathcal{G}$ from K, replace *every* edge of K with L. Thus if



Suppose that f is a 3-multicoloring of $\mathcal{G}$. By (2.4.11), f must be in fact a 3-coloring of $\mathcal{G}$, that is, each vertex of $\mathcal{G}$ has exactly one color. Now color each vertex of K with the color of its identified terminal in $\mathcal{G}$. Since the terminals of L must have different colors, K must be properly 3-colored.

Now suppose that f is a 3-coloring of K. Color the terminals of $\mathcal{G}$ by the colors of their identified vertices in K, and color the non-terminals of $\mathcal{G}$ as indicated in (2.4.12), permuting colors appropriately. The result is clearly an ordinary 3-coloring of $\mathcal{G}$. Since each L in $\mathcal{G}$ is properly directly 3-multicolored, the only possible way for this coloring of $\mathcal{G}$ to be an improper direct 3-multicoloring is for a counterexample path to have a terminal as one of its two interior vertices. But the neighborhood of a terminal vertex v in $\mathcal{G}$ looks like:



It is now easily seen that no counterexample path exists. Thus $\mathcal{G}$ can be properly directly 3-multicolored. □

2.4.5. Other Complexity Results

Two remarks are in order about the last two proofs. First, since 3GCP is NP-Complete even for planar K, and since the edge replacement graphs L are themselves planar in both cases, finding optimum PSimDCs or GSimDCs for Hessians whose sparsity patterns correspond to planar graphs is NP-Complete. Though this fact has little practical significance in itself, it does have an interesting corollary. It is well-known that a planar graph on n vertices can have at most $3n - 6 = O(n)$ edges (see Bondy and Murty (1976), Corollary 9.5.2). Thus the intractability of finding optimal PSimDCs and GSimDCs is not due to requiring algorithms to process nearly dense Hessians. In particular, it is also NP-Complete to find optimal PSimDCs and GSimDCs for Hessians with $O(n)$ unknowns (with density $O(1/n)$).

Second, the proof of Theorem 2.4.4 shows that for every graph $\mathcal{G}$ resulting from the reduction, any GSimDC for the corresponding Hessian which has only three groups must in fact be a PSimDC. Thus the proof of Theorem 2.4.4 also proves Theorem 2.4.3 as a corollary. However, since the proof of Theorem 2.4.3 given is quite simple and is a useful warm-up for the proof of Theorem 2.4.4, it was included despite its technical redundancy.

The complexity of substitution methods can also be analyzed through graph coloring. Coleman and Moré (1982) consider a particular subclass of substitution methods called **lower-triangular substitution methods** defined originally in Powell and Toint (1979). These methods bear roughly the analogous relation to general substitution methods as SeqDCs do to general direct covers. Coleman and Moré show that finding an optimal set of difference directions for a lower-triangular substitution method is equivalent to a certain kind of graph coloring that they call **triangular coloring**, and use this equivalence to prove the following (see their Theorem 7.2).

Theorem 2.4.9: Finding an optimal lower-triangular substitution set of difference directions is NP-Complete. □

2.4.6. Heuristic Approaches to Direct Methods

The NP-Completeness theorems in this section are rather discouraging, since the conventional wisdom is that NP-Completeness is tantamount to intractability. On a more positive note, much work has been done on finding near-optimal, polynomial-time, heuristic algorithms for NP-Complete problems (see Garey and Johnson (1979), chapter 6).

In the present case, the most obvious heuristic approach is to reduce D2GCP to GCP and then apply known heuristic results on GCP to the reduced graph. Given a graph $\mathcal{G} = (V, E)$, define $D_2(\mathcal{G})$ (the **distance-2 completion** of $\mathcal{G}$) to be the graph on the same vertex set V, and with edges $E = \{\{i,j\} \mid i \text{ and } j \text{ are distance 2 or less apart in } \mathcal{G}\}$. Equivalently, when the vertex-vertex adjacency matrix A of $\mathcal{G}$ has a non-zero diagonal, then $D_2(\mathcal{G})$ is the graph whose adjacency matrix is A^2. A third equivalent formulation is that $D_2(\mathcal{G})$ is the **intersection graph** (see Golumbic (1980), Section 1.2) of the columns of its adjacency matrix. It is easy to verify that a coloring of V is a proper distance-2 coloring of $\mathcal{G}$ if and only if it is a proper (distance-1) coloring of $D_2(\mathcal{G})$ (note that this reduction also implies that D1GCP is NP-Complete).

If there were a "good" heuristic for GCP, it could be composed with $D_2(\bullet)$ to obtain a "good" heuristic for D2GCP. Coleman and Moré (1981), Section 4, gives a good overview of the present state of the art in GCP heuristics, which is *not* "good". In fact, if $c^H(\mathcal{G})$ denotes the number of colors used by the best known heuristic on graph $\mathcal{G}$, and $\chi(\mathcal{G})$ denotes the optimal number of colors necessary for $\mathcal{G}$ (its **chromatic number**), then in the worst case

$$\max_{\mathcal{G} \text{ on } n \text{ vertices}} \frac{c^H(\mathcal{G})}{\chi(\mathcal{G})} = O(n^\alpha), \tag{2.4.13}$$

where $\alpha = 1 - \frac{1}{\chi(\mathcal{G})-1}$ (the best known heuristic and the bound (2.4.13) are due to Widgerson (1982); see also Johnson (1974) for worst case analysis of other graph coloring heuristics). Two facts mitigate the severity of (2.4.13). First, the range of $D_2(\bullet)$ does not include all graphs, and hence a better bound than (2.4.13) can be obtained for D2GCP. Second, average-case results have been obtained for GCP heuristics that are considerably better than (2.4.13).

To improve on (2.4.13) for D2GCP, consider the specific heuristic called the **distance-2 sequential algorithm (D2SA)**. Define $\mathcal{N}(i) = \{j \neq i \mid j \text{ is distance } \leq 2 \text{ from } i\}$, the **distance-2 neighborhood** of a vertex i in a graph. Thus, if i has color c in a proper distance-2 coloring, no $j \in \mathcal{N}(i)$ can be color c. Then D2SA assigns color

$$\min\{c \geq 1 \mid \text{no } j \in \mathcal{N}(i),\ j < i, \text{ is colored } c\}$$

to vertex i, $i = 1, \ldots, |V|$. That is, D2SA assigns vertex i the smallest color not conflicting with those already assigned. (D2SA is just the distance-2 version of the best known GCP heuristic, the **sequential algorithm**, which is called the **CPR** method in its applications to approximating sparse Jacobians; see Curtis, Powell and Reid (1974).) Let $c^S(\mathcal{G})$ denote the number of colors used by D2SA when applied to $\mathcal{G}$.

In order to obtain bounds on $c^S(\mathcal{G})$, two definitions are required. The **maximum degree** of $\mathcal{G}$, $\Delta(\mathcal{G})$, is defined as

$$\Delta(\mathcal{G}) = \max_i |\{j \mid \{i,j\} \in E(\mathcal{G})\}|.$$

The **distance-2 chromatic number** of $\mathcal{G}$, $\chi_2(\mathcal{G})$, is defined as the optimal number of colors in a proper distance-2 coloring of $\mathcal{G}$, *i.e.*,

$$\chi_2(\mathcal{G}) = \min\{k \mid \mathcal{G} \text{ has a proper distance-2 coloring with } k \text{ colors}\}.$$

The following theorem bounds $\chi_2(\mathcal{G})$ and $c^S(\mathcal{G})$ in terms of $\Delta(\mathcal{G})$, and a corollary improves (2.4.13) for D2SA:

Theorem 2.4.10: Let $d = \Delta(\mathcal{G})$. Then

$$d + 1 \leq \chi_2(\mathcal{G}) \leq c^S(\mathcal{G}) \leq d^2 + 1 \tag{2.4.14}$$

for all graphs $\mathcal{G}$.

Proof: Let i be a vertex incident to exactly d edges, and note that i and its d nearest neighbors must all be different colors in a proper distance-2 coloring; this proves the lower bound in (2.4.14). The second inequality in (2.4.14) is trivial.

To prove the upper bound in (2.4.14), note that for any vertex i, $|\mathcal{N}(i)| \leq d + d(d-1) = d^2$. Suppose that D2SA assigns color l to vertex i; by definition of D2SA, color l is assigned only if at *least one vertex* of each color $1, \ldots, l-1$ is in $\mathcal{N}(i)$. Thus, if i were assigned color $l > d^2 + 1$, then $|\mathcal{N}(i)| \geq d^2 + 1$ (a contradiction). (This proof is essentially a constructive proof of Corollary 8.2.1 in Bondy and Murty (1976).) □

Corollary 2.4.11: For all $n \geq 1$,

$$\max_{\mathcal{G} \text{ on } n \text{ vertices}} \frac{c^S(\mathcal{G})}{\chi_2(\mathcal{G})} \leq \sqrt{n-1} + 1 = O(\sqrt{n}). \tag{2.4.15}$$

Proof: Clearly, $c^S(\mathcal{G}) \leq n$. Let $k = \chi_2(\mathcal{G})$. Applying the first and third inequalities of (2.4.14) yields

$$c^S(\mathcal{G}) \leq (k-1)^2 + 1. \tag{2.4.16}$$

Consider two cases:
Case 1: If $n \leq (k-1)^2 + 1$, then $\sqrt{n-1} + 1 \leq k$ and so

$$\frac{c^S(\mathcal{G})}{k} \leq \frac{n}{k} \leq \frac{n}{\sqrt{n-1}+1} = \sqrt{n-1} + 1 - 2 + \frac{2}{\sqrt{n-1}+1} \leq \sqrt{n-1} + 1.$$

Case 2: If $n > (k-1)^2 + 1$, then $k < \sqrt{n-1} + 1$, and so

$$\frac{c^S(\mathcal{G})}{k} \leq \frac{(k-1)^2 + 1}{k} = k - 2 + \frac{2}{k} \leq k < \sqrt{n-1} + 1.$$

(Corollary 2.4.11 is essentially (4.6) of Coleman and Moré (1981) in the special case that the matrix is square and symmetric). □

Graphs that attain bound (2.4.16) for a certain ordering of their vertices exist for $k = 1, 2, 3, 4$. The cases $k = 1, 2$ are trivial. For $k = 3$, consider $\mathcal{G}_3 = (V_3, E_3)$ defined by

$$V_3 = x_{ij} \qquad i = 1, 2, 3, \quad j = 1, 2, 3, 4, 5,$$

$$E_3 = \{ x_{ij}, x_{i+1,j+1} \} \qquad \text{all } i, j \qquad \text{(subscripts modulo 3 and 5).}$$

Then D2SA assigns x_{ij} color i when the vertices are ordered by i (which is optimal by (2.4.14)), and assigns x_{ij} color j when the vertices are ordered by j (which is the worst possible, by (2.4.16)). For $k = 4$, consider $\mathcal{G}_4 = (V_4, E_4)$ defined by

$$V_4 = x_{ij} \qquad i = 1, 2, 3, 4, \quad j = 1, \ldots, 10,$$

$$E_4 = \left\{ \begin{array}{ll} \{ x_{ij}, x_{i+2,j+5} \} & \text{all } j \\ \{ x_{ij}, x_{i+1,j+2} \} & \text{all odd } j \\ \{ x_{ij}, x_{i+1,j+4} \} & \text{all even } j \end{array} \right\} \quad \text{all } i \qquad \text{(subscripts modulo 4 and 10).}$$

Then D2SA applied to $\mathcal{G}_4$ also colors x_{ij} with i when ordered by i, and with j when ordered by j (which are again respectively optimal and worst possible).

Extending this construction seems to be extremely difficult. Its extension appears to be roughly equivalent to solving a hard open problem in extremal graph theory (see Bollabás (1978), Section IV.1). Even if it could be extended, the number of vertices is given by $n = k\big((k-1)^2 + 1\big)$, so that

$$\frac{c^S(\mathcal{G}_k)}{k} = O(n^{1/3}), \tag{2.4.17}$$

which is a better result than (2.4.15). Thus, while (2.4.14), (2.4.15) and (2.4.16) are better results than (2.4.13), we conjecture that (2.4.17) is also a (better) bound.

Turning from the worst case to the average case, Grimmet and McDiarmid (1975) proved the following theorem:

Theorem 2.4.12: Fix n vertices, and let vertices i and j be independently connected by an edge with fixed probability p, $0 < p < 1$. Let $c^C(\mathcal{G})$ be the number of colors used by CPR on $\mathcal{G}$, and $\chi(\mathcal{G})$ be the optimal number of colors (so that $c^C(\mathcal{G})$ and $\chi(\mathcal{G})$ are random variables). Then

$$\frac{c^C(\mathcal{G})}{\chi(\mathcal{G})} \leq 2 + \epsilon$$

for all $\epsilon > 0$ with probability $1 - o(1)$. □

Thus, on average, CPR almost never performs more than twice as badly as the optimal strategy. This theorem has at least two unsatisfactory features in this context. First, sparsity patterns in practical problems are not uniformly random as assumed in Theorem 2.4.12. Second, even if they were, the density of sparsity patterns tends to be $O(1/n)$ rather than constant with increasing n. It would be useful to determine a better random model for sparsity patterns, or at least to prove Theorem 2.4.12 under the assumption that $p = O(1/n)$.

The weakness of such random models in predicting practical performance is illustrated by the computational experiments of Coleman and Moré (1982) with a version of a CPR heuristic, namely CPR applied to the sparsity pattern with the columns in the **smallest-last ordering** (see Coleman and Moré (1981) for details). In their Table 4.1, column "maxr" represents a lower bound on $\chi_2(\mathcal{G})$, and column "sl" represents $c^S(\mathcal{G})$, using the smallest-last ordering. When averaged over the 30 real examples that they tested, $c^S(\mathcal{G})$ used 14.43 colors whereas the lower bound averaged 13.6 colors. Thus the improved CPR used less than one extra color above the optimal on average, and $c^S(\mathcal{G})$ averaged at most 6.1% larger than $\chi_2(\mathcal{G})$, a big improvement over Theorem 2.4.12.

2.5. Lower Bounding Elimination Methods

The computational experiments reported in Coleman and Moré (1982), Tables 4.1 and 8.1 are quite intriguing. For the 30 practical problems on which they tested various heuristics, the best NDC heuristic used 14.43 groups on average, the best SeqDC heuristic used 11.63 groups and the best lower-triangular substitution heuristic used 7.87 groups. This progression leads to speculation about the minimum possible number of difference directions for a given sparsity pattern, and whether there is a polynomial algorithm to compute it. The corresponding problem for sparse Jacobians is relatively easy; see Newsam and Ramsdell (1982), Theorem 3.

The ultimate lower bound on the number of difference directions necessary to approximate a given H, call it $\gamma(H)$, is the minimum number needed by an elimination method, since these methods allow complete freedom in choosing the d^l. This section presents some results that give various lower bounds for $\gamma(H)$, and presents some evidence that the best lower bound is polynomially computable. It is conjectured that the best lower bound is tight for every H.

It must be emphasized at the outset that elimination methods are studied here *not* because they are claimed to be in any sense practical. Instead the aim is to formulate a procedure whereby it can be easily checked how far the substitution heuristics are from $\gamma(H)$. If these heuristics are found to be close to $\gamma(H)$ on average, the practicality of the heuristics evidently makes further work on elimination methods in practice unappealing. Alternatively, if there is to be a significant gap between the substitution heuristics and $\gamma(H)$, it would be justified to investigate whether there are practically implementable elimination heuristics which out-perform the substitution heuristics.

In this section we shall *not* assume that the sparsity pattern of H has a non-zero diagonal. Also, H and its associated graph will be referred to interchangeably, so that it will make sense to write that H is bipartite. For case of referral equations (2.2.1) are reproduced here, deleting the hat on H and the dependence on x^0 for simplicity:

$$H d^l = \Delta^l, \quad l = 1, 2, \ldots, k. \tag{2.5.1}$$

(Recall that Δ^l is defined as $(g(x^0 + d^l) - g(x^0))^T$.) Note that (2.5.1) is a set of nk linear equations in the $\binom{n+1}{2}$ unknowns $h_{11}, h_{21}, h_{22}, h_{31}, \ldots, h_{n1}, h_{n2}, \ldots, h_{nn}$. Denote the coefficient matrix of (2.5.1) by $A^{n,k}$.

2.5.1. A General Lower Bound on Evaluations

In the situation of interest, sparsity causes many of the unknowns to be deleted, that is, causes many of the columns in $A^{n,k}$ to be deleted. However, assume for the moment that H is completely dense. Then by simple linear algebra, the largest number of unknowns that can be solved for by a subsystem of equations (2.5.1) is equal to the rank of $A^{n,k}$. Suppose that it can be shown that the maximum rank possible for $A^{n,k}$ is, say, r_{nk}. Note that r_{nk} would have to be increasing in k. As sparsity comes into play, columns are eliminated from $A^{n,k}$, and its rank can only decrease. Thus, even for a sparse H, the maximum number of unknowns that can be solved for is still at most r_{nk}. By this reasoning a lower bound on $\gamma(H)$ can be calculated as follows. Denote the number of unknowns in H by η, and the smallest k such that $r_{nk} \geq \eta$ by k^*. Then at least k^* evaluations are necessary to approximate H. Thus we shall now focus our attention on determining r_{nk}.

The rank of $A^{n,k}$ is affected by the numerical values in the d^l. By assuming that the d^l satisfy the **Haar condition**, namely that every square submatrix of the matrix whose l^{th} column is d^l is nonsingular, the rank of $A^{n,k}$ is maximized. (The Haar condition is implied by the assumption that the entries of the d^l are independent algebraic indeterminates. It is also implied by the assumption that the d^l are perturbed from their given values by infinitesimals, similar to the construction often used in non-degeneracy proofs.) For example, choosing the d^l as columns from a Vandermonde matrix (see Knuth (1973), Section 1.2.3, exercises 36–45) satisfies the Haar condition.

To determine r_{nk}, we investigate the structure of $A^{n,k}$. Label the i^{th} row of the set of equations associated with Δ^l by "Δ^l_i," and label the column corresponding to variable h_{ij} by "h_{ij}." For $n = 4$ and $k = 2$, $A^{4,2}$ has the form

$$
\begin{array}{c|cc|ccc|cccc}
 & h_{11} & h_{21}\ h_{22} & h_{31} & h_{32} & h_{33} & h_{41} & h_{42} & h_{43} & h_{44} \\
\Delta^1_1 & d^1_1 & d^1_2 & d^1_3 & & & d^1_4 & & & \\
\Delta^2_1 & d^2_1 & d^2_2 & d^2_3 & & & d^2_4 & & & \\
\hline
\Delta^1_2 & & d^1_1\ d^1_2 & & d^1_3 & & & d^1_4 & & \\
\Delta^2_2 & & d^2_1\ d^2_2 & & d^2_3 & & & d^2_4 & & \\
\hline
\Delta^1_3 & & & d^1_1 & d^1_2 & d^1_3 & & & d^1_4 & \\
\Delta^2_3 & & & d^2_1 & d^2_2 & d^2_3 & & & d^2_4 & \\
\hline
\Delta^1_4 & & & & & & d^1_1 & d^1_2 & d^1_3 & d^1_4 \\
\Delta^2_4 & & & & & & d^2_1 & d^2_2 & d^2_3 & d^2_4
\end{array}
\qquad (2.5.2)
$$

In general the entry in row Δ^p_q, column h_{ij} of $A^{n,k}$ is

$$
\begin{cases}
d^p_j & \text{if } q = i, \\
d^p_i & \text{if } q = j, \\
0 & \text{otherwise.}
\end{cases}
\qquad (2.5.3)
$$

Besides yielding a lower bound on $\gamma(H)$, the determination of r_{nk} is also interesting from another point of view. Each additional gradient evaluation yields n more seemingly independent linear equations. Because of symmetry there are only $\binom{n+1}{2} = \frac{n(n+1)}{2}$ unknowns in a dense Hessian. Thus it might appear that only $\lceil \frac{n+1}{2} \rceil$ evaluations would suffice to approximate a dense Hessian, since the number of equations $= n \cdot \lceil \frac{n+1}{2} \rceil \geq \binom{n+1}{2} = \eta$. Perhaps then the number of gradient evaluations needed even for a dense Hessian could be reduced below n by a clever choice of difference directions. However, the next theorem shows that such savings are not possible. The theorem appears to be well-known in the folklore, but we know of no published proof.

Theorem 2.5.1: The maximum number of unknowns that can be determined by a set of gradient evaluations along any k directions, $0 \leq k \leq n$ is given by

$$\begin{aligned} r_{nk} &= n + (n-1) + \cdots + (n-k+1) \\ &= nk - \binom{k}{2} = \binom{n+1}{2} - \binom{n-k+1}{2}. \end{aligned} \tag{2.5.4}$$

In particular, n evaluations are necessary to obtain all $\binom{n+1}{2}$ unknowns in the completely dense case. Bound (2.5.4) is sharp for some sparsity patterns.

We shall give two different proofs of Theorem 2.5.1, the first a column-oriented proof, the second a row-oriented proof. The column-oriented proof is more direct since it exhibits an explicit subset of r_{nk} of the unknowns which form a basis, but the row-oriented proof is simpler.

Column-oriented Proof of Theorem 2.5.1: Partition the rows of $A^{n,k}$ as in (2.5.2) into n **row blocks** of k equations each, the i^{th} row block consisting of equations $\Delta_i^1, \Delta_i^2, \ldots, \Delta_i^k$. Partition the columns of $A^{n,k}$ as in (2.5.2) into n **column blocks** of $1, 2, \ldots, n$ unknowns each, the j^{th} column block consisting of columns $h_{j1}, h_{j2}, \ldots, h_{jj}$. Let $A_{i,j}^{n,k}$ denote the i,j^{th} submatrix of this partition. To simplify notation, define c^j as the k-vector of the j^{th} components of the $\{d^i\}$, i.e., $c^j = (d_j^1, d_j^2, \ldots, d_j^k)^T$, $j = 1, \ldots, n$. Then (2.5.2) and (2.5.3) imply that each $A_{i,j}^{n,k}$ is completely described by

$$A_{i,j}^{n,k} = \begin{cases} 0, & \text{if } i > j; \\ (c^1, c^2, \ldots, c^j), & \text{if } i = j; \\ (0, 0, \ldots, \underset{i}{c^j}, \ldots, \underset{j}{0}), & \text{if } i < j. \end{cases} \tag{2.5.5}$$

To complete the proof, it must be shown that $\operatorname{rank}(A^{n,k}) = r_{n,k}$. Let E be the set of columns $h_{i,j}$ with $k < j \leq i < n$, and let F be the complementary set of columns. We shall show that the columns of F can be used to eliminate the columns of E. Note that $|F| = r_{n,k}$, and that each column in F involves only c^i with $i \leq k$

Define λ^l to be the solution of the system

$$(c^1 \quad c^2 \quad \cdots \quad c^k)\lambda^l = -c^l, \qquad l = k+1, k+2, \ldots, n$$

(λ^l exists and is unique under the assumption of the Haar condition). The following computations show that linear combinations of the columns in F, using the $\{\lambda^l\}$ as multipliers, can be used to eliminate the columns in E; since the form of the linear combinations is complicated, the result is best understood by referring to the following example (2.5.6), which is (2.5.2) re-written in the new notation with the multipliers defined below for the case that $k = 2$, $q = 4$ and $p = 3$:

h_{11}	h_{21}	h_{22}	h_{31}	h_{32}	h_{33}	h_{41}	h_{42}	h_{43}	h_{44}
$\lambda_1^4\lambda_1^4$	$\lambda_2^4\lambda_1^4$	$\lambda_2^4\lambda_2^4$				λ_1^4	λ_2^4		1
$\lambda_1^4\lambda_1^3 + \lambda_1^3\lambda_1^4$	$\lambda_2^4\lambda_1^3 + \lambda_2^3\lambda_1^4$	$\lambda_2^4\lambda_2^3 + \lambda_2^3\lambda_2^4$	λ_1^4	λ_2^4		λ_1^3	λ_2^3	1	

$$\left(\begin{array}{c|cc|ccc|cccc} c^1 & c^2 & & c^3 & & & c^4 & & & \\ \hline & c^1 & c^2 & & c^3 & & & c^4 & & \\ \hline & & & c^1 & c^2 & c^3 & & & c^4 & \\ \hline & & & & & & c^1 & c^2 & c^3 & c^4 \end{array}\right) \tag{2.5.6}$$

Let p and q satisfy $k < p < q \leq n$, so that $h_{q,q}$ and $h_{q,p}$ are typical columns in E. To eliminate column $h_{q,q}$ from $A^{n,k}$, add to it λ_j^q times column $h_{q,j}$, $j = 1, \ldots, k$, and $\lambda_i^q\lambda_j^q$ times column $h_{i,j}$, $i = 1, \ldots, k$,

$j = 1, \ldots, i$ (these multipliers are the first line of λ's in (2.5.6)). From (2.5.5) and (2.5.6) the resulting column is zero in row blocks i, $k < i \leq n$, $i \neq q$, since no column with a non-zero coefficient is non-zero in these row blocks. In row block i, $1 \leq i \leq k$, the non-zero contributions to the resulting column are $\sum_{l \leq i} \lambda_i^q \lambda_l^q c^l$ from column block i, $\lambda_i^q \lambda_l^q c^l$ from column block l, $i < l \leq k$, and $\lambda_i^q c^q$ from column block q, for a total of

$$\lambda^q \left(\sum_{l \leq k} \lambda_l^q c^l + c^q \right) = 0, \qquad i = 1, 2, \ldots, k.$$

In row block q, the only non-zero contribution is from column block q, which is

$$\sum_{l \leq k} \lambda_l^q c^l + c^q = 0.$$

Thus the resultant column is zero and column $h_{q,q}$ is indeed dependent on the columns of F.

Now column $h_{q,p}$ is eliminated using the columns in F. Add to it λ_j^p times column $h_{q,j}$, $j = 1, \ldots, k$, λ_j^q times column $h_{p,j}$, $j = 1, \ldots, k$, and $(\lambda_i^q \lambda_j^p + \lambda_i^p \lambda_j^q)$ times column $h_{i,j}$, $i = 1, \ldots, k$, $j = 1, \ldots, i$ (these multipliers are shown in the second row of λ's in (2.5.6)). There is no non-zero contribution to the resultant column in any column block i, $k < i \leq n$, $i \neq p, q$. In row block i, $1 \leq i \leq k$, there is a contribution of $\sum_{l \leq i} (\lambda_i^q \lambda_l^p + \lambda_i^p \lambda_l^q) c^l$ from column block i, a contribution of $(\lambda_i^q \lambda_l^p + \lambda_i^p \lambda_l^q) c^l$ from column block l, $i < l \leq k$, a contribution of $\lambda_i^q c^p$ from column block p, and a contribution of $\lambda_i^p c^q$ from column block q, for a total of

$$\lambda_i^q \left(\sum_{l \leq k} \lambda_l^p c^l + c^p \right) + \lambda_i^p \left(\sum_{l \leq k} \lambda_l^q c^l + c^q \right) = 0, \qquad i = 1, 2, \ldots, k.$$

In row block p, there is a contribution of $\sum_{l \leq k} \lambda_l^q c^l$ from column block p and a contribution of c^q from column block q for a total of

$$\sum_{l \leq k} \lambda_l^q c^l + c^q = 0.$$

In row block q, the only non-zero contribution is from column block q and is

$$\sum_{l \leq k} \lambda_l^p c^l + c^p = 0.$$

Once again the row block totals are all zero, so that column $h_{q,p}$ is also dependent on the columns in F.

Eliminating the $1 + 2 + \cdots + (n - k)$ columns of E shows that $\text{rank}(A^{n,k}) \leq r_{n,k}$. To show that $\text{rank}(A^{n,k}) = r_{n,k}$, delete the columns of E from $A^{n,k}$, and delete the last $k - i$ rows from each row block i, $i < k$. The remaining matrix is $r_{n,k}$ by $r_{n,k}$ and is block upper triangular with square, non-singular diagonal blocks. Thus this submatrix of $A^{n,k}$ is non-singular, and hence $\text{rank}(A^{n,k}) \geq r_{n,k}$.

To show that this bound is sharp, consider the sparsity pattern which has $h_{i,j}$ as an unknown whenever $i \leq k$ or $j \leq k$. By letting d^i be the i^{th} unit vector for $i = 1, 2, \ldots, k$, all $r_{n,k}$ of the unknowns can clearly be solved for, and thus the bound of the theorem is attained. □

Row-oriented Proof of Theorem 2.5.1: This proof is due to Hoffman (1982).

For each 2-subset $\{i, j\} \subseteq \{1, 2, \ldots, k\}$ define an nk-vector z^{ij} with entries indexed with the same labels as the rows of $A^{n,k}$ by

$$z_p^{ij} = \begin{cases} d_q^j & \text{if } p = \Delta_q^i, \\ -d_q^i & \text{if } p = \Delta_q^j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5.7}$$

We now show that z^{ij} is in the null space of the columns of $A^{n,k}$, *i.e.*, $z^{ij} A^{n,k} = 0$. Take columns h_{11} and h_{21} of $A^{n,k}$ as representative examples. By (2.5.3), column h_{11} is non-zero only in rows Δ_1^l, $l = 1, 2, \ldots, k$. Comparing with (2.5.7), z^{ij} and column h_{11} are both non-zero only in rows Δ_1^i and Δ_1^j. In row Δ_j^i, z^{ij} is d_1^j and column h_{11} is d_1^i, and in row Δ_1^j, z^{ij} is $-d_1^i$ and column h_{11} is d_1^j, so that the value of the product is $d_1^j d_1^i - d_1^i d_1^j = 0$, as desired.

Similarly, by (2.5.3) and (2.5.7) the only non-zero contributions to the product for column h_{21} are

from row	Δ_1^i	Δ_2^i	Δ_1^j	Δ_2^j
in z^{ij}	d_1^j	d_2^j	$-d_1^i$	$-d_2^i$
in column h_{21}	d_2^i	d_1^i	d_2^j	d_1^j

which again totals zero, as desired.

The next claim is that the z^{ij} are independent. Let Z be the $nk \times \binom{k}{2}$ matrix with columns indexed by 2-subsets of $\{1,2,\ldots,k\}$ whose $\{i,j\}^{\text{th}}$ column is z^{ij}. It is necessary to show that Z has full rank. Set $d^l = e^l$, the l^{th} unit vector, so that $d_p^l = 1$ if $p = l$, and is 0 otherwise. Choosing d^l in this way could only decrease the rank of Z. Let $\{i,j\}$ be a 2-subset of $\{1,2,\ldots,k\}$ and consider row Δ_j^i of Z. From (2.5.7), entry $\{i,j\}$ of row Δ_j^i of Z is ± 1, and every other entry is zero. Thus the submatrix of Z consisting of rows Δ_j^i for all subsets $\{i,j\}$ is diagonal, and Z has full rank.

The rank of the null space of the rows of $A^{n,k}$ is therefore at least the number of columns of Z, namely $\binom{k}{2}$, and so the rank of $A^{n,k}$ can be at most $nk - \binom{k}{2}$. To show that rank $A^{n,k} \geq r_{nk}$, consider the sparsity pattern

$$\begin{pmatrix} C & D \\ D^T & 0 \end{pmatrix},$$

where C is a $k \times k$ dense matrix, and D is an $k \times (n-k)$ dense matrix. It has r_{nk} unknowns. By choosing the d^l as above, all of its unknowns are clearly determined by k evaluations, and consequently the corresponding set of equations must have rank at least r_{nk}. But the set of equations arising from such a sparsity pattern has a coefficient matrix which is a submatrix of $A^{n,k}$, and so rank $A^{n,k} = r_{nk}$. □

2.5.2. A Bipartite Lower Bound on Evaluations

In considering sparsity patterns with some (or even all) zero diagonal entries, it is possible to obtain a sharper lower bound than that of Theorem 2.5.1, by considering **bipartite** sparsity patterns, *i.e.*, sparsity patterns whose associated graphs are bipartite.

A sparsity pattern is bipartite if and only if it has a principal permutation so that its structure looks like

$$\begin{pmatrix} 0 & C \\ C^T & 0 \end{pmatrix}, \tag{2.5.8}$$

where C is an $s \times t$ matrix. Such a Hessian can clearly be approximated by at most $\min(s,t)$ gradient evaluations, by differencing along either the first s or the last t unit vectors.

When the matrix C in (2.5.8) is completely dense, call the coefficient matrix of the equations (2.5.1) B^k. As was the case with $A^{n,k}$, the maximum number of unknowns that can be determined by k evaluations of a sparse bipartite Hessian is bounded above by the rank of B^k. The next theorem is the bipartite analogue of Theorem 2.5.1.

Theorem 2.5.2: The maximum number of unknowns of a sparse bipartite Hessian (as in (2.5.8)) that can be determined from a set of gradient evaluations along any k directions, $0 \leq k \leq \min(s,t)$, is

$$r_{stk} = (s+t)k - k^2.$$

In particular, when the matrix C in (2.5.8) is completely dense, $\min(s,t)$ evaluations are needed to obtain all st unknowns. This bound is sharp for some sparsity patterns.

Proof: This proof uses the same ideas as the row-oriented proof of Theorem 2.5.1. We shall show that rank $B^k = r_{stk}$.

Denote the first s indices of H by $S1, S2, \ldots, Ss$, and the last t indices by $T1, T2, \ldots, Tt$. The columns of B^k are labelled with h_{SiTj} for $i = 1,2,\ldots,s$ and $j = 1,2,\ldots,t$, and the rows with Δ_p^l, $p = S1, S2, \ldots, Ss, T1, T2, \ldots, Tt$, $l = 1,2,\ldots,k$. The entry in row Δ_p^l, column h_{SiTj} of B^k corresponding to

(2.5.3) equals

$$\begin{cases} d^l_{Tj} & \text{if } p = Si, \\ d^l_{Si} & \text{if } p = Tj, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5.9}$$

For every ordered pair (i,j) with $i,j \in \{1,2,\ldots,k\}$, define an nk-vector z^{ij} indexed as the rows of B^k by

$$z^{ij}_p = \begin{cases} d^j_{Sq} & \text{if } p = \Delta^i_{Sq} \text{ and } i \le j, \\ -d^j_{Tq} & \text{if } p = \Delta^i_{Tq} \text{ and } i \ge j, \\ -d^i_{Tq} & \text{if } p = \Delta^j_{Tq} \text{ and } i \le j, \\ d^i_{Sq} & \text{if } p = \Delta^j_{Sq} \text{ and } i \ge j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5.10}$$

Note that this definition is consistent when $i = j$.

The first claim is that $z^{ij}B^k = 0$, which is verified for a typical column h_{SuTv}. Assume without loss of generality that $i \le j$. From (2.5.9) and (2.5.10) the contributions to the product are

from row	Δ^i_{Su}	Δ^j_{Tv}
in z^{ij}	d^j_{Su}	$-d^i_{Tv}$
in column h_{SuTv}	d^i_{Tv}	d^j_{Su}

and the total product is zero, as claimed.

The second claim is that the z^{ij} are independent. As before, let Z be the $nk \times k^2$ matrix whose columns are the z^{ij}, and set $d^l = e^{Sl} + e^{Tl}$, so that $d^l_p = 1$ if $p = Sl$ or Tl, and is 0 otherwise. Choosing these d^l could only decrease the rank of Z. Let $i,j \in \{1,2,\ldots,k\}$ where $i \le j$, say, and consider row Δ^i_{Sj} of Z. From (2.5.10) column z^{ij} is $+1$ in row Δ^i_{Sj}, and every other entry is zero. If $i > j$ consider row Δ^i_{Tj}, where column z^{ij} is -1 and all other entries are zero. Since this subset of rows picks out a diagonal submatrix of Z, it has full rank

Because Z has k^2 columns, rank $B^k \le nk - k^2$. Now consider the Hessian with bipartite sparsity pattern

$$\begin{pmatrix} 0 & & C & D \\ & & E & 0 \\ C^T & E^T & 0 & \\ D^T & 0 & & \end{pmatrix},$$

where C is dense and $k \times k$, D is dense and $k \times (t-k)$, and E is dense and $(s-k) \times k$. It has $k(s+t) - k^2$ unknowns. All of its unknowns can be approximated with only k evaluations by using the d^l defined above. By the same reasoning as in the proof of Theorem 2.5.1, it follows that rank $B^k = nk - k^2$. □

2.5.3. Examples of Lower Bounding

We now give some examples of how Theorems 2.5.1 and 2.5.2 are used to calculate lower bounds for $\gamma(H)$. First consider **dense band matrices**, which have an unknown in entry i,j if and only if $|i-j| \le w$; w is called the **bandwidth** of the matrix. For instance, when $n = 5$ and $w = 3$, the sparsity pattern looks like

$$\begin{pmatrix} \times & \times & \times & 0 & 0 \\ \times & \times & \times & \times & 0 \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix}.$$

Note that such a matrix has r_{nw} unknowns. It is well-known (see Coleman and Moré (1981), Theorem 5.1) that dense band matrices can be approximated by using the difference directions

$$d^i = \sum_{l=1}^{\lfloor \frac{n-i}{w} \rfloor} e^{i+lw}, \quad i = 1,2,\ldots,w.$$

In fact, these d^i correspond to a SeqDC for dense band matrices. Since the number of evaluations equals the smallest k for which the number of unknowns is at most r_{nk}, by Theorem 2.5.1 these sets of directions have the minimum possible cardinality for dense band matrices. Thus the bound of Theorem 2.5.1 is actually achieved for dense band matrices.

The bound of Theorem 2.5.1 can also be attained for the **complete graphs without loops**, which have unknowns at every entry i, j, except when $i = j$. The insights discussed here are due to Hoffman (1982). For instance, when $n = 3$, the sparsity pattern is

$$\begin{pmatrix} 0 & \times & \times \\ \times & 0 & \times \\ \times & \times & 0 \end{pmatrix}. \tag{2.5.11}$$

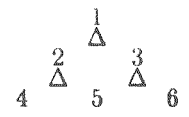
It is known (see Powell and Toint (1979), equation (5.1)) that by differencing along $d^1 = (1, 1, 1)$, (2.5.11) can be approximated in only one evaluation. (Approximating the matrix (2.5.11) with this d is an elimination method, and it is easy to see that any substitution method must use more than one evaluation. The matrix (2.5.11) seems to be the only example known, a point which is further explored later in this section.)

Let H^n have the incidence matrix of the complete graph without loops on n vertices as its sparsity pattern; it has $\binom{n}{2}$ unknowns. Suppose that H^n can by optimally approximated by γ_n gradient evaluations by an as yet unknown elimination method. Then $\gamma_{n+1} \leq \gamma_n + 1$ since H^{n+1} can surely be approximated by first differencing along $d^1 = e^{n+1}$ to get the last row and column of H^{n+1}, and then using the γ_n approximation scheme on the remaining unknowns, which have the same sparsity pattern as H^n.

Let λ_n be the lower bound on γ_n implied by Theorem 2.5.1, so that $\lambda_n = \min\{ k \mid \binom{n}{2} \leq nk - \binom{k}{2} \}$ which implies that $\lambda_n = \lceil \frac{1+2n-\sqrt{8n+1}}{2} \rceil$. Since (2.5.11) is H^3, by the remarks above and the fact that $\lambda_n \leq \gamma_n$, the first few values of λ_n and γ_n are

n	1	2	3	4	5	6
λ	0	1	1	2	3	3
γ	0	1	1	2	3	?

We now show that $\gamma_6 = 3$. Arrange the vertices of H^6 in an array like:

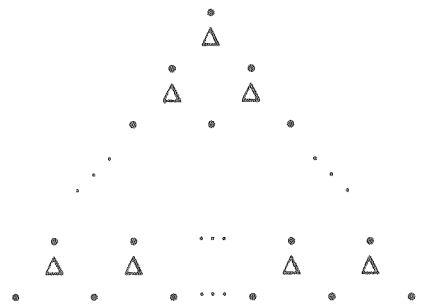


and difference along the three indicated triangles, *i.e.* along $d^1 = (1, 1, 1, 0, 0, 0)$, $d^2 = (0, 1, 0, 1, 1, 0)$ and $d^3 = (0, 0, 1, 0, 1, 1)$. By (2.5.11) these directions determine all the edges of H^6 in the triangles, *i.e.* edges 12, 13, 23, 24,..., 56. The 123 triangle difference (d^1) gives $h_{15} + h_{25} + h_{35} = \Delta_5^1$ in row 5. Edges 25 and 35 are triangle edges, and so this equation determines edge 15 (and by symmetry, edges 34 and 26 are also determined). In row 4, the 123 triangle equation is $h_{14} + h_{24} + h_{34} = \Delta_4^1$. Edge 24 is a triangle edge, and it was shown above that edge 34 can also be determined; thus edge 14 (and so also 16 and 46) is determined. But all the edges in H^6 have now been determined, which implies that $\gamma_6 = 3$.

For $n > 6$ the inductive method that showed that $\gamma_{n+1} \leq \gamma_n + 1$ can be used as long as $\lambda_{n+1} = \lambda_n + 1$. It can be shown that $\lambda_{n+1} \neq \lambda_n + 1$ fails if and only if $\frac{1+2n-\sqrt{8n+1}}{2}$ is integral. This holds if and only if $8n + 1$ is a perfect odd square, which happens if and only if $n = \binom{k+1}{2}$ for some integer k. After $n = 6$, the next such n is $n = 10 = \binom{4+1}{2}$. When n has such a value, $\lambda_n = \lambda_{n-1} = \binom{k}{2}$, which is the next lower such n. Such integers are known as **triangular** numbers for the reason that when n is triangular, n points can be arranged in a triangular array similar to the configuration for $n = 6$ above.

Now it is inductively easy to show that when $n = \binom{k+1}{2}$, $\lambda_n = \gamma_n = \binom{k}{2}$ by using the $\binom{k}{2}$ triangular

differences



The induction hypothesis shows that every edge of H^n is determined by these differences except for those involving a vertex of the largest triangle and a point on the opposite edge. But these can be determined from the other edges by a process similar to that described above for $n = 6$. Thus the triangular numbers are doubly triangular in this context.

For values of n that are not triangular numbers, both λ_n and γ_n increase by one. Thus for all n, $\lambda_n = \gamma_n$, and so once again the implicit bound of Theorem 2.5.1 is attained.

For the next example, consider the sparsity pattern

$$\begin{pmatrix} & & & \times & \times & \times \\ & 0 & & \times & \times & \times \\ & & & \times & \times & \times \\ \times & \times & \times & & & \\ \times & \times & \times & & 0 & \\ \times & \times & \times & & & \end{pmatrix}, \tag{2.5.12}$$

for which $n = 6$ and $\eta = 9$. Theorem 2.5.1 says that two evaluations can determine as many as $6 \cdot 2 - \binom{2}{2} = 11$ unknowns, and thus gives a lower bound of only 2. But since (2.5.12) is also a (complete) bipartite graph with $s = t = 3$, Theorem 2.5.2 gives a better lower bound of 3 (and also says that 3 evaluations are optimal. Example (2.5.12) illustrates that Theorem 2.5.2 can imply a higher bound than Theorem 2.5.1.

There is a way to sharpen further the bounds of Theorems 2.5.1 and 2.5.2. Consider the sparsity pattern

$$\begin{pmatrix} \times & \times & \times & 0 \\ \times & \times & \times & 0 \\ \times & \times & \times & 0 \\ 0 & 0 & 0 & \times \end{pmatrix}, \tag{2.5.13}$$

with $n = 4$ and $\eta = 7$. When $k = 2$, Theorem 2.5.1 concludes that as many as 7 unknowns could be calculated. But in approximating (2.5.13), the leading 3×3 submatrix must be approximated as well, and Theorem 2.5.1 implies that $k = 3$ evaluations are necessary to approximate a dense 3×3 matrix. Thus a better bound can be obtained in some cases from a submatrix than from the whole matrix.

For any (non-empty) $S \subseteq \{1, 2, \ldots, n\}$ the bound of Theorem 2.5.1 can be computed based on the submatrix of H whose rows and columns are in S. The largest lower bound so computed is then a possibly sharper lower bound for $\gamma(H)$ than the bound based on the whole matrix.

If a submatrix of H corresponds to a bipartite graph, a lower bound should be computed using the sharper Theorem 2.5.2. Unfortunately, sometimes Theorem 2.5.2 can give a sharper result even when a submatrix does not correspond to a bipartite graph. Consider the sparsity pattern

$$\begin{pmatrix}
\times & 0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\
0 & \times & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\
0 & 0 & \times & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times \\
0 & 0 & 0 & \times & 0 & 0 & \times & \times & \times & \times & \times & \times \\
0 & 0 & 0 & 0 & \times & 0 & \times & \times & \times & \times & \times & \times \\
0 & 0 & 0 & 0 & 0 & \times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times & 0 & 0 & 0 & 0 & 0 \\
\times & \times & \times & \times & \times & \times & 0 & \times & 0 & 0 & 0 & 0 \\
\times & \times & \times & \times & \times & \times & 0 & 0 & \times & 0 & 0 & 0 \\
\times & \times & \times & \times & \times & \times & 0 & 0 & 0 & \times & 0 & 0 \\
\times & \times & \times & \times & \times & \times & 0 & 0 & 0 & 0 & \times & 0 \\
\times & \times & \times & \times & \times & \times & 0 & 0 & 0 & 0 & 0 & \times
\end{pmatrix}, \tag{2.5.14}$$

with $n = 12$ and $\eta = 48$. Since $5 \cdot 12 - \binom{5}{2} = 50 > 48$, Theorem 2.5.1 gives a best lower bound of $k = 5$ (even if checked for all submatrices). But by deleting the diagonal entries, the matrix becomes bipartite and Theorem 2.5.2 gives the better lower bound of $k = 6$.

Thus, to achieve the best lower bound derivable from Theorems 2.5.1 and 2.5.2, it is necessary to consider not just the vertex-induced subgraphs of H (which correspond to submatrices of H), but all the edge-induced subgraphs as well. The edge-induced submatrices must be checked for bipartiteness to see whether Theorem 2.5.2 can give a higher bound.

2.5.4. Computing Lower Bounds in Theory

Next we consider how to compute the bounds implicit in Theorems 2.5.1 and 2.5.2. In principle, as was shown in the proofs of Theorem 2.5.1, $\gamma(H)$ can be computed by the following algorithm.

Algorithm γ:

0. Set $k = 1$.
1. Construct the matrix $A^{n,k}$, using, say, columns of a Vandermonde matrix for the d^l.
2. Delete the columns of $A^{n,k}$ corresponding to entries of H known to be zero, yielding $\overline{A}^{n,k}$.
3. Calculate $\text{rank}(\overline{A}^{n,k}) = t_k$. If $t_k \geq \eta$, then $\gamma(H) = k$. Stop.
4. Otherwise, set $k \leftarrow k + 1$ and go to 1.

With infinite-precision arithmetic, Algorithm γ is a polynomial algorithm, since Step 3 can be performed at most n times on a matrix whose size is bounded by n^2. Calculating rank is an $O(\text{size}^3)$ operation, yielding a total time bound of $O(n^7)$. However in practice, with finite-precision arithmetic, the decision in Step 3 is not clear cut. Calculating the rank of *any* numerical matrix is extremely difficult in practice (see Peters and Wilkinson (1970)), so much so that the conventional wisdom among numerical analysts is that numerical rank cannot be precisely defined. Even if it could be, the well-known classes of matrices guaranteed to satisfy the Haar condition (such as the Vandermonde matrices) are notoriously ill-conditioned and difficult to work with (see Newsam and Ramsdell (1981), p. 13 for similar concerns in the context of Jacobian approximation). Thus Algorithm γ would not be practical even if rank were calculable.

A more practical implementation of Algorithm γ is to use randomly generated vectors for the d^l in Step 1. With probability nearly 1, a random matrix satisfies the Haar condition, and in practice $\overline{A}^{n,k}$ is usually well enough conditioned that Step 3 can be carried out satisfactorily in finite-precision arithmetic. By repeating the randomized Algorithm γ several times using different random d^l, a high degree of confidence in the answer can be obtained. Indeed just such an algorithm has been implemented (in order to search for a counterexample to a conjecture that comes later), and it has performed satisfactorily on small problems with $n < 12$.

The randomized Algorithm γ has some drawbacks. First, it becomes increasingly slow to compute the LU-factorization that is used to calculate the rank of the $nk \times \eta$ matrices. Second, one has less and less faith in the computed answer as n gets large, due to the usual numerical difficulties in computing rank mentioned above, coupled with a smaller degree of confidence that the random matrices satisfy the Haar condition to the tolerance of the computer. Third, the whole procedure is esthetically unsatisfying to mathematical

sensibilities because the number $\gamma(H)$ is an intrinsic characteristic of H, and calculating ranks of random matrices seems to be a roundabout way of computing it. A purely combinatorial algorithm would seem much more appropriate.

Unfortunately such an algorithm does not now exist. However, we make the following two conjectures:

Conjecture 2.5.3: $\gamma(H)$ is *equal* to the largest lower bound computable via subgraphs using Theorems 2.5.1 and 2.5.2.

Conjecture 2.5.4: There is a purely combinatorial polynomial algorithm for computing $\gamma(H)$ that is fast in practice.

The evidence for Conjecture 2.5.3 is that $\gamma(H)$ has been computed for many small H ($n \leq 12$) by using the randomized Algorithm γ, and no counterexample has been found. Also, the conjecture is true for all the examples discussed above. That is, every H tried so far does have some subgraph K such that $\gamma(H)$ is the smallest k for which the bound of either Theorem 2.5.1 or 2.5.2 is satisfied on K.

The evidence for Conjecture 2.5.4 is stronger than a lack of counterexamples. There already exists an almost practically effective way to compute $\gamma(H)$, namely the randomized version of Algorithm γ. The fact that $\gamma(H)$ can be easily computed for many H appears to be inconsistent with any supposition that finding $\gamma(H)$ is NP-Complete. A striking characteristic of most NP-Complete problems is that they are no easier in practice than they are in theory. Thus the existence of the randomized Algorithm γ seems strong evidence for Conjecture 2.5.4.

2.5.5. Computing Lower Bounds in Practice

If Conjectures 2.5.3 and 2.5.4 are both true, it would follow that there is a (practical) combinatorial polynomial algorithm for computing $\lambda(H)$, the largest lower bound implied by Theorems 2.5.1 and 2.5.2 over all subgraphs of H. Some preliminary work is presented next on how to compute $\lambda(H)$, which can also be taken as evidence for both conjectures.

For simplicity, assume at first that H is bipartite; then all of its edge-induced subgraphs are also bipartite, so that only the bound of Theorem 2.5.2 is relevant. Let E be any subset of the unknowns of H (or of the edges of its graph), and let $N(E) = \{\, i \mid h_{ij} \in E, \text{ some } j \,\}$. For example, if

$$H = \begin{pmatrix} 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ \times & \times & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \\ \times & \times & 0 & 0 & 0 \end{pmatrix}$$

and $E = \{\, h_{13}, h_{14}, h_{15} \,\}$, then $N(E) = \{\, 1, 3, 4, 5 \,\}$.

In the subgraph of H determined by E, Theorem 2.5.2 says that k evaluations might suffice for the $|E|$ unknowns if $|E| \leq |N(E)| \cdot k - k^2$. Thus the largest lower bound over all E must be

$$\min\{\, k \mid |E| \leq |N(E)| \cdot k - k^2 \quad \text{for all } E \,\}. \tag{2.5.15}$$

To see why, denote the minimizing k in (2.5.15) by k^*, and let E^* be an edge subset that blocks k^* from being smaller, so that $k^* = \min\{\, k \mid |E^*| \leq |N(E^*)| \cdot k - k^2 \,\}$. But this is the definition of the lower bound on $\gamma(H)$ derivable from E^*, so that k^* is a lower bound on $\gamma(H)$. For any other edge subset E, $|E| \leq |N(E)| \cdot k^* - (k^*)^2$, which implies that the lower bound derivable from E is at most k^*.

Equation (2.5.15) is reminiscent of Phillip Hall-type theorems (see, *e.g.*, Welsh (1976), pp. 97–98). Let B be a bipartite graph with left vertices S and right vertices T. For $U \subseteq S$, let $\Gamma(U) = \{\, i \in T \mid \{\, s, i \,\}$ is an edge of B for some $s \in U \,\}$. The Phillip Hall theorem of interest is

Theorem 2.5.5: Let $m = |S|$ and $d \geq 0$. Then B has a matching of size $m - d$ if and only if

$$|U| \leq |\Gamma(U)| + d, \quad \text{for all } U \subseteq S. \tag{2.5.16}$$

(See Welsh (1976), Theorem 7.4.) □

Theorem 2.5.5 is interesting because it provides a polynomial algorithm that simultaneously verifies an exponential number of inequalities. That is, verifying the 2^m inequalities in (2.5.16) is equivalent by Theorem 2.5.5 to finding out whether B has a matching of size at least $m - d$; since good polynomial algorithms exist for finding a maximum cardinality matching (see Lawler (1976), Chapter 5), the equivalence implies that there is a polynomial algorithm for verifying the inequalities (2.5.16).

It is simple to generalize Theorem 2.5.5 slightly when the inequalities to be verified are

$$|U| \leq |\Gamma(U)| \cdot k + d \quad \text{for all } U \subseteq S. \tag{2.5.17}$$

Recall that the maximum cardinality matching problem on B is equivalent to a network flow problem N as follows (see Lawler (1976), Section 5.2). Direct each S,T edge from S to T with capacity ∞, add an arc (s,i) with capacity 1 for all $i \in S$, and add an arc (j,t) with capacity 1 for all $j \in T$. Then N has a flow of value $m - d$ if and only if B has a matching of size $m - d$.

Now change the capacity of each (j,t) arc from 1 to k (which corresponds to "multiplying" each T vertex of B by k). Theorem 2.5.5 becomes

Theorem 2.5.6: The 2^m inequalities (2.5.17) are true if and only if N has a flow of value $m - d$. □

Since network flow also has a polynomial algorithm (see Lawler (1976), Chapter 4), there is a fast way of verifying (2.5.17) as well.

Now suppose that the $2^m - 1$ inequalities

$$|U| \leq |\Gamma(U)| \cdot k \quad d \quad \text{for all } \emptyset \neq U \subseteq S \tag{2.5.18}$$

could also be verified in polynomial time for any $d \geq 0$. Consider the bipartite graph B which has $S = \{\text{unknowns of } H\}$, $T = \{1, 2, \ldots, n\}$ and edge $\{h_{ij}, l\}$ when $l = i$ or j. Then for $E \subseteq S$, $\Gamma(E) = N(E)$. Hence, by setting $d = k^2$ and iterating the hypothetical procedure for $k = n, n-1, \ldots$, the minimizing k in (2.5.15) could be determined. This line of reasoning makes it interesting to find a polynomial algorithm to solve (2.5.18) (an apparently minor variant of (2.5.17)).

Such an algorithm has been provided by Saks and Kahn (1983). Consider the network N of Theorem 2.5.6; let N_i denote N with the capacity of the single arc (s,i) changed from 1 to ∞. By the usual argument, a minimum cut for N_i must be of the form $\{s\} \cup U \cup \Gamma(U)$ for some $U \subseteq S$. Since the capacity of (s,i) is ∞, i must be in U. Thus the minimum cut, and so the maximum flow, for N_i solves the problem

$$\min_{i \in U \subseteq S} m \quad |U| + k \cdot |\Gamma(U)|. \tag{2.5.19}$$

Solve each of the maximum flow/minimum cut problems N_i and let U^* be a minimizer in (2.5.19) for an N_i with the *smallest* capacity minimum cut. Now, if (2.5.18) is satisfied, then certainly $m + d \leq |\Gamma(U^*)| \cdot k - |U^*| + m =$ smallest value of any N_i flow. Conversely, suppose that

$$m + d \leq |\Gamma(U^*)| \cdot k - |U^*| + m. \tag{2.5.20}$$

By definition of U^*, the right-hand side of (2.5.20) is less than or equal to $|\Gamma(U)| \cdot k - |U| + m$ for all $U \subseteq S$, and so (2.5.18) is satisfied.

The problem of verifying the $2^m - 1$ inequalities in (2.5.18) has been reduced to solving m network flow problems, and so can be done in polynomial time. Let V be the set of nodes of N_i, and E the set of its arcs. Then the complexity of solving one N_i maximum flow problem is $O(|V||E|\log|V|)$ (see Papadimitriou and Stieglitz (1982), Chapter 9). Since verifying (2.5.18) involves m problems, it is of complexity $O(m|V||E|\log|V|)$.

For the problem of interest, $|S| = m = \eta$, $|V| = |S \cup T| = n + \eta$, and $|E| = n + 3\eta$. The procedure iterates at most η times, for a total time complexity of $O(\eta(n+\eta)^2 \log(n+\eta))$. When $\eta = O(n)$, the complexity reduces to $O(n^3 \log n)$.

To re-capitulate, we have shown that when H is bipartite, the following algorithm *exactly* calculates $\lambda(H)$:

Algorithm B:

0. Set $k = \lfloor \frac{n}{2} \rfloor - 1$.
1. For $i =$ each unknown in H do
   Construct network N_i.
   Solve maximum flow on N_i, with value v_i.
2. Let $v^* = \min_i v_i$.
3. If $\eta + k^2 \leq v^*$ set $k \leftarrow k - 1$, go to 1.
4. Else answer $\lambda(H) = k + 1$.

The existence of Algorithm B is certainly consistent with the truth of Conjectures 2.5.3 and 2.5.4. When H is *not* bipartite, it is easy to see that the best bound given by Theorem 2.5.1 can be calculated by replacing Step 3 of Algorithm B by "If $\eta + \binom{k}{2} \leq v^* \ldots$," since the only difference between the conclusions of Theorems 2.5.1 and 2.5.2 is the substitution of $\binom{k}{2}$ for k^2. Unfortunately, as example (2.5.14) shows, the answer resulting from the modified Algorithm B is not in general equal to $\lambda(H)$, even when all the diagonal elements of H are unknowns. Resolving this difficulty is an area for more research.

2.5.6. A Bound for Higher-Order Derivatives

A question that naturally arises is how this research extends to approximating higher-order derivatives. Such an extension is not a practical concern, for storing and working with a moderately large order-3 derivative array, even if it is sparse, would be prohibitively expensive. Nevertheless, a mathematical sense of completion can make such extensions interesting. The urge to generalize has been resisted in most of the rest of this thesis (except perhaps in the first proof of Theorem 2.4.1), but we shall yield to it here and indicate how to generalize Theorem 2.5.1 to higher-order derivatives.

Let us review the definition of a higher-order derivative. For a function $F: \mathbb{R}^n \to \mathbb{R}$, its derivative of order m, $\nabla^m F$, evaluated at point x^0, is the $n \times n \times \cdots \times n$ (m times) array of numbers

$$\nabla^m F(x^0) = \left(h_{i_1,i_2,\ldots,i_m}\right), \quad \text{where} \quad h_{i_1,i_2,\ldots,i_m} = \frac{\partial^m F(x^0)}{\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_m}}.$$

By symmetry of repeated derivatives, $h_{i_1,i_2,\ldots,i_m} = h_{\pi(i_1),\pi(i_2),\ldots,\pi(i_m)}$ for any m-permutation π. Thus, the number of potentially different entries in $\nabla^m F(x^0)$ is the cardinality of the set $I_{nm} = \{ (i_1, i_2, \ldots, i_m) \mid 1 \leq i_1 \leq i_2 \leq \cdots \leq i_m \leq n \}$, the set of **$m$-selections** from $\{ 1, 2, \ldots, n \}$ (they are selections instead of subsets since they can have repeated entries).

Let $P_{kl} = \{ (i_1, i_2, \ldots, i_l) \mid 1 \leq i_1 < i_2 < \cdots < i_l \leq k \}$, the set of **$l$-subsets** of $\{ 1, 2, \ldots, k \}$, so that $|P_{kl}| = \binom{k}{l}$. Then the bijection between elements of I_{nm} and $P_{n+m-1,m}$ given by

$$(i_1, i_2, \ldots, i_m) \in I_{nm} \leftrightarrow (i_1, i_2 + 1, \ldots, i_m + m - 1) \in P_{n+m-1,m}$$

(see Knuth (1973), Section 1.2.6, exercise 60) shows that $|I_{nm}| = \binom{n+m-1}{m}$, so that the number of different unknowns in a dense $\nabla^m F(x^0)$ is also $\binom{n+m-1}{m}$ (a special case is that when $m = 2$, the number of unknowns in a dense Hessian is $\binom{n+1}{2}$, as already discussed).

Let $H^m = \nabla^m F(x^0)$. In approximating H^m by finite-differencing $\nabla^{m-1} F$ along directions d^l, the approximation $\hat{H}^m$ satisfies the linear equations

$$\sum_{i_1=1}^{n} \hat{h}_{i_1,i_2,\ldots,i_m} d^l_{i_1} = \left(\nabla^{m-1} F(x^0 + d^l) - \nabla^{m-1} F(x^0) \right)_{i_2,i_3,\ldots,i_m}, \tag{2.5.21}$$

$1 \leq i_j \leq n$, $1 \leq l \leq k$. Each d^l would appear to give rise to n^{m-1} equations in (2.5.21), namely one for each different $i_2, i_3, \ldots, i_m$, but since $\nabla^{m-1} F$ is also symmetric, these equations are identical under permutation of $i_2, i_3, \ldots, i_m$. Thus it can be assumed without loss of generality that $i_2 \leq i_3 \leq \cdots \leq i_m$, so that each d^l actually gives rise to $I_{n,m-1}$ equations in (2.5.21). It follows that (2.5.21) is a set of $k\binom{n+m-2}{m-1}$ equations in $\binom{n+m-1}{m}$ unknowns.

The usual approximation method would again choose $d^l = e^l$. It is easy to see that k evaluations of $\nabla^{m-1}F$ with these d^l (directly) determines every entry of $\hat{H}^m$ whose smallest subscript is at most k. If an entry of $\hat{H}^m$ has all its subscripts greater than k, then its subscripts are effectively an m-selection from the set $\{k+1, k+2, \ldots, n\}$ of cardinality $n-k$. Thus $|I_{n-k,m}| = \binom{n-k+m-1}{m}$ out of the $\binom{n+m-1}{m}$ total unknowns are *not* determined by the k evaluations along the unit vectors, so that $\binom{n+m-1}{m} - \binom{n-k+m-1}{m}$ unknowns *are* determined. The proof of Theorem 2.5.1 leads to the conjecture that at most this number of unknowns can be determined by general d^l as well, and indeed the following theorem is the proper generalization of Theorem 2.5.1.

Theorem 2.5.8: When approximating $\nabla^m F$ (possibly with sparsity conditions) by finite-differencing $\nabla^{m-1}F$ along directions $d^1, d^2, \ldots, d^k$, at most

$$r_{nmk} = \binom{n+m-1}{m} - \binom{n-k+m-1}{m}$$

unknowns can be determined by the k evaluations of $\nabla^{m-1}F$.

In particular, n evaluations are necessary to approximate $\nabla^m F$ when it is dense. This bound is tight for some sparsity patterns.

Sketch of Proof: This proof is very much in the spirit of the row-oriented proof of Theorem 2.5.1.

Let A^{nmk} be the $k\binom{n+m-2}{m-1} \times \binom{n+m-1}{m}$ coefficient matrix of equations (2.5.21). As was the case in Theorem 2.5.1, it suffices to show that rank $A^{nmk} = r_{nmk}$ when $\hat{H}^m$ is completely dense. A key observation in the proof is the identity

$$\binom{n+m-1}{m} - \binom{n-k+m-1}{m} = -\sum_{i=1}^{m}(-1)^i \binom{n+m-i-1}{m-i}\binom{k}{i}, \tag{2.5.22}$$

of which (2.5.4) is a specialization. This identity is easy to prove by induction.

A sequence of matrices $A^{nmk} = Z^1, Z^2, \ldots, Z^m$ can be defined with entries from the d^l, where Z^i is $\binom{n+m-i-1}{m-i}\binom{k}{i} \times \binom{n+m-i}{m-i+1}\binom{k}{i-1}$. Note that the row size of Z^i is the absolute value of the i^{th} term of (2.5.22), and that the column size of Z^i equals the row size of Z^{i-1}. The property that the Z^i are constructed to satisfy is that $Z^i Z^{i-1} = 0$, $i = 2, 3, \ldots, m$, so that each row of Z^i is in the null space of the columns of Z^{i-1}.

Now set $d^l = e^l$, which can only decrease the rank of each Z^i. As the base of an induction, using these particular d^l it can be shown that Z^m has full row rank, namely $\binom{k}{m}$. Thus, since Z^{m-1} has $n\binom{k}{m-1}$ rows and Z^m is in its null space, rank $Z^{m-1} \leq n\binom{k}{m-1} - \binom{k}{m}$. But with these d^l a square, diagonal submatrix of Z^{m-1} of size $n\binom{k}{m-1} - \binom{k}{m}$ can be found, and hence rank $Z^{m-1} = n\binom{k}{m-1} - \binom{k}{m}$, the last two terms of (2.5.22).

At the general step it is known that rank $Z^{i+1} = -\sum_{l=i+1}^{m}(-1)^{l-i}\binom{n+m-l-1}{m-l}\binom{k}{l}$ and Z^i has $\binom{n+m-i-1}{m-i}\binom{k}{i}$ rows, so that

$$\text{rank } Z^i \leq \sum_{l=i}^{m}(-1)^{l-i}\binom{n+m-l-1}{m-l}\binom{k}{l}. \tag{2.5.23}$$

But then there is a square submatrix of Z^i of size (2.5.23) which is diagonal with this choice of d^l, hence (2.5.23) is really an equality, and the induction can proceed. The induction terminates at $i = 1$ which yields

$$\begin{aligned}\text{rank } Z^1 = \text{rank } A^{nmk} &= -\sum_{i=1}^{m}(-1)^i\binom{n+m-i-1}{m-i}\binom{k}{i} \\ &= \binom{n+m-1}{m} - \binom{n-k+m-1}{m}\end{aligned}$$

as desired. □

It would be mathematically interesting to obtain a similar generalization of Theorem 2.5.2.

2.6. Reflections on Sparse Hessians

We have investigated the approximation of sparse Hessians, largely from the point of view of computational complexity. Sections 2.2 and 2.3 showed that various methods that have been proposed for approximating sparse Hessian can be classified in a helpful way.

Using this classification, Section 2.4 showed that finding an optimal approximation scheme for each subclass of direct methods is an NP-Complete problem. The theorem of Coleman and Moré (1982), which is quoted as Theorem 2.4.9, gives a similar NP-Completeness result for a subclass of substitution methods. Their theorem suggests that as with direct methods, all subclasses of substitution methods are NP-Complete. Fully investigating the complexity of substitution methods is a useful topic for future research.

By contrast, the results of Section 2.5 tend to support the view that determining the minimum number of gradient evaluations needed by a given sparsity pattern for an elimination method can be computed by a polynomial algorithm (however, computing a set of numerically "reasonable" difference directions that realizes the minimum may be much harder). Thus in passing through the spectrum of approximation methods from direct methods (simple, stable, large number of evaluations) to elimination methods (complicated, possibly unstable, smallest number of evaluations), a boundary between NP-Completeness and polynomial algorithms seems to be crossed between the substitution methods and the elimination methods. Clearly more work needs to be done to establish the truth or falsity of Conjectures 2.5.3 and 2.5.4.

An intriguing additional reason to study Conjectures 2.5.3 and 2.5.4 is that very few examples are known of sparse Hessians where an optimal elimination method uses strictly fewer gradient evaluations than an optimal substitution method. The standard (and apparently, essentially the only) example of this phenomenon is (2.5.11) (though it is likely that all complete graphs without loops also fall in this class). It is not clear whether such examples are inherently rare, or whether there has been insufficient work in constructing them. If such examples are rare, then establishing the truth or falsity of Conjectures 2.5.3 and 2.5.4 is even more important since the ability to compute $\gamma(H)$ for a substitution method would be a valuable guide for a substitution heuristic. On the other hand, if such examples are common, being able to compute $\gamma(H)$ might aid in searching for them.

Though practicalities have been mentioned along the way, our emphasis has been on complexity rather than computation. Thus the reader may still be uncertain as to what method to choose to approximate a sparse Hessian. The direct methods reported in Coleman and Moré (1982) are simple, numerically stable, and very fast (both in finding the groups and in approximating H given the groups), and empirically give fairly good results (see their Table 4.1). Substitution methods are inherently less numerically stable than direct methods, though Powell and Toint (1979) show that the accumulated error in a substitution method cannot grow too fast. The triangular substitution methods in Coleman and Moré (1982) are almost as simple as their direct methods, reasonably numerically stable, fast in finding the groups, somewhat slower in solving for H given the groups, but empirically use significantly fewer gradient evaluations than their direct heuristics (compare their Tables 4.1 and 8.1). To our knowledge, no general elimination methods have been proposed. Except in special cases, such as the complete graphs without loops mentioned in Section 2.5 (which effectively use a substitution method except for solving systems like (2.5.11)), elimination methods are expected to have such potentially unreliable numerical properties as to make them practically useless. At this point, the triangular substitution heuristics in Coleman and Moré (1982) seem to be the best for general usage.

This chapter has resolved many of the previously open questions about approximating sparse Hessians. However, much work remains before the subject is completely understood.

Chapter 3

Making Sparse Matrices Sparser

3.1. Introduction to Making Matrices Sparser

Many large-scale constrained optimization problems are of the form

$$\begin{aligned} &\min F(x) \\ &\text{s. t.} \quad Ax = b \\ &l \leq x \leq u, \end{aligned} \tag{3.1.1}$$

where l, u, $x \in \mathrm{R}^n$, $F: \mathrm{R}^n \rightarrow \mathrm{R}$ and A is an $m \times n$ matrix. This is a linearly constrained problem with bounded variables. Usually m is less than n, and hence there are many x that satisfy $Ax = b$.

Quite large problems of the form (3.1.1) have been solved, some with $m > 10,000$, $n > 50,000$ (see, *e.g.*, Hillier and Lieberman (1974), pp. 180–181). If such an A were dense, then storing and accessing its entries would cause an optimization program to be painfully slow.

The reason that very large problems can be solved in practice is that they are **sparse**; most of the entries of A are zero. A rule of thumb that is used in some applications is that an average column of A usually has less than ten non-zeros in it, often less than five non-zeros. A matrix with the above dimensions would be expected to have only about 500,000 non-zeros, a decrease of three orders of magnitude from the number of possible entries.

Define the **density** of a sparse matrix as the fraction of entries of A that are non-zero. Then when $n = O(m)$, the number of possible entries of real-life matrices is $O(m^2)$ whereas their number of non-zeros is only $O(n)$, so that their density is $O(1/m)$.

To take advantage of sparsity it is necessary to store i and j along with a_{ij}, thus incurring a storage overhead, but a relatively small one (matrix indices can often be stored in many fewer bits than numerical values). The necessity of manipulating the indices makes some simple sparse matrix tasks quite complicated. For example, it can be non-trivial to transpose a sparse matrix in some representations (see Gustavson (1973)). Programs for processing sparse matrices are therefore much longer, more complicated and harder to develop than for their dense counterparts.

Nevertheless the immense savings in execution time over comparable dense algorithms warrants taking account of sparsity. By exploiting sparsity, much larger problems have been solved much faster than they have would been otherwise. Indeed, the largest problems could not effectively be solved at all without sparse matrix techniques.

Sparse methods are not faster than dense methods simply because there are many fewer numbers to keep track of. There is another, more subtle phenomenon working in favor of sparsity. Consider solving the system of linear equations

$$Bx = b \tag{3.1.2}$$

when B is $m \times m$ and non-singular. The usual dense Gaussian elimination procedure takes $O(m^3)$ time. Since the number of entries of B is m^2, the execution time is superlinear in the amount of data.

Now consider solving the same system when B is sparse, assuming $O(m)$ non-zeros. It has been empirically observed that a well-implemented sparse Gaussian elimination technique takes only $O(m)$ time (see Duff (1977), Table 3). This observation is true partly because even real problems without apparent structure seem to have some hidden structure, though in a way that has resisted quantification. It seems doubtful that such good results would be obtained on matrices whose non-zero entries were randomly located. Thus in a typical sparse situation, linear equations can be solved in time linear in the amount of data. Since solving linear equations is a ubiquitous operation in optimization, such an improvement represents a significant speed-up compared to the dense case.

Since the time required to perform many kinds of sparse matrix operations is proportional to the number of non-zeros in A, trying to make the given sparse linear constraints $Ax = b$ even sparser seems to be a natural problem to solve. That is, instead of accepting the degree of sparsity in the model as formulated, it might help to *increase* the degree of sparsity (decrease the density). More succinctly, since sparsity is a virtue, sparser should be better.

An obvious application of such an effort is that in solving (3.1.1), optimization routines solve many systems of linear equations like (3.1.2), where B varies over various submatrices of A (see Gill, Murray and Wright (1981), Chapter 5). Thus, if A were sparser, the various B's would (on average) be sparser. Since execution time depends on number of non-zeros, the speed of optimization would increase.

A less obvious application is when the linear constraints of (3.1.1) are replaced by **non-linear** constraints $c(x) = 0$, where c is a function from R^n to R^m. Such non-linearly constrained problems are often sparse in the sense that the Jacobian of $c(x)$ is a sparse matrix, and its **sparsity pattern** (zero/non-zero structure) is independent of x. An algorithm that is able to make a matrix sparser using only its sparsity pattern could therefore be useful for non-linear problems.

These possible applications lead to considering the

Sparsity Problem (SP): Given

$$Ax = b, \tag{3.1.3}$$

find an equivalent system

$$\hat{A}x = \hat{b} \tag{3.1.4}$$

which is as sparse as possible, where **equivalent** means that the same set of x's satisfy both systems.

From simple linear algebra, (3.1.3) and (3.1.4) are equivalent if and only if $\hat{A} = TA$ and $\hat{b} = Tb$ for some $m \times m$ non-singular matrix T. Thus, solving SP is equivalent to finding a T that minimizes the number of non-zeros in TA. This chapter explores some ways to solve SP in theory and in practice.

3.1.1. Relationship to Bipartite Matching

The methods that we shall use to solve SP involve **bipartite matching** theory. There is a simple correspondence between bipartite graphs and sparsity patterns of rectangular matrices. When we write a sparsity pattern a zero is represented by "0" or a blank, and a non-zero by "×". Given the sparse matrix A, define the bipartite graph B by setting the left nodes of $B = \{\text{rows of } A\}$, the right nodes of $B = \{\text{columns of } A\}$, and the edges of $B = \{\{i,j\} \mid a_{ij} \neq 0\}$. For example, if

$$A = \begin{pmatrix} \times & \times & 0 \\ 0 & \times & \times \end{pmatrix}, \quad \text{then } B = \quad \begin{array}{ll} \textcircled{1} & \textcircled{1} \\ \textcircled{2} & \textcircled{2} \\ & \textcircled{3} \end{array}$$

This correspondence allows us refer to sparsity patterns and bipartite graphs interchangeably. In this chapter sparsity patterns will be displayed as matrices, but the language of bipartite graphs will be used to describe them.

A subset P of the non-zeros of A such that no two elements of P lie in the same row or column is classically known as a **partial transversal** (see Welsh (1976), Section 7.1). A partial transversal corresponds to a (not necessarily maximum) **matching** (see, *e.g.*, Lawler (1976), Chapter 5) in a bipartite graph (*i.e.*, a subset of edges with no common vertices). For example, the circled transversal corresponds to the heavy matching in the bipartite graph B:

$$A = \begin{pmatrix} \otimes & \times & 0 \\ 0 & \otimes & 0 \\ \times & 0 & \otimes \end{pmatrix}, \quad B = \quad \begin{array}{ll} \textcircled{1} & \textcircled{1} \\ \textcircled{2} & \textcircled{2} \\ \textcircled{3} & \textcircled{3} \end{array}$$

We shall favor the term "matching" even though it is historically inappropriate for matrices.

A matching in A is called **row-perfect** if all rows of A are in the matching; **column-perfect** is defined similarly. A matching is **perfect** if it is both row- and column-perfect. A **maximum matching** is one with a maximum number of non-zeros. If $R \subseteq \{1, 2, \ldots, m\}$ and $C \subseteq \{1, 2, \ldots, n\}$ then A_{RC} denotes the submatrix of A indexed by rows in R and columns in C. Let $\mathcal{E}(A) = \{\text{non-zeros of } A\}$, and let $M(A_{RC})$ be the size of a maximum matching in A_{RC}; $M(A_{RC})$ is sometimes called the **term rank** of A_{RC} (see Ryser (1963), Chapter 5). An important property of maximum matchings that will be used repeatedly is stated in the following proposition.

Proposition 3.1.1: If M is a maximum matching in A, the rows and columns of A can be permuted so that A can be partitioned as

$$A = \begin{pmatrix} C & 0 \\ D & E \end{pmatrix}, \tag{3.1.5}$$

where $M \subseteq \mathcal{E}(C) \cup \mathcal{E}(D)$; $M \cap \mathcal{E}(C)$ is column-perfect for C and is row-perfect for C if and only if M is row-perfect; and $M \cap \mathcal{E}(E)$ is row-perfect for E and is column-perfect for E if and only if M is column-perfect. □

This proposition follows from the König-Egervary Theorem (see Ryser (1963), Theorem 5.1). As an example of the proposition, consider

$$A = \begin{pmatrix} C & 0 \\ D & E \end{pmatrix} = \left(\begin{array}{ccc|ccc} \otimes & \times & & & & \\ & \otimes & & & & \\ \hline \times & \times & \otimes & & \times & \\ \times & & \times & \otimes & \times & \end{array}\right),$$

with the circled maximum matching. The matching is not column-perfect for E, nor for A, but it is row-perfect for both C and A.

3.1.2. Possible Approaches to Increasing Sparsity

Two possible assumptions can be made in dealing with sparse matrix problems like SP. The first is that almost all the information about A is contained in its sparsity pattern, and almost none is embodied in the actual values of the non-zeros. This assumption is used by the graph models of the location of fill-in during sparse Gaussian elimination which occur in theorems about the complexity of minimizing fill-in (see Problem GT46] in Garey and Johnson (1979)).

The complementary assumption is that the non-zero values have a structure that can be exploited in solving SP. This assumption would lead to an algorithm that would try to discover numerical relations among the non-zeros in an effort to increase sparsity. An example of this assumption as used for a different problem is the work of Bixby and Cunningham (1983) on solving linear programs faster by finding large embedded networks.

We shall use the first approach in this chapter. Indeed, in the application to non-linear constraints discussed above, no other approach is possible since there is a fixed sparsity pattern with changing numeric entries.

3.1.3. Overview of this Chapter

Section 3.2 opens with a discussion of why SP is difficult without making a generality assumption. A rigorous definition of the assumption used in the rest of the chapter is then given, and is applied to derive polynomial algorithms to solve SP. With the assumption, a polynomial algorithm is constructed that solves an important subproblem of SP, the One Row Sparsity Problem. This algorithm is at the heart of all the other algorithms.

In Section 3.3, the One Row Algorithm of Section 3.2 is used to derive two polynomial algorithms for solving SP. One of the algorithms is important for theoretical reasons, and the other can be modified into a practically implementable algorithm. Some theoretical consequences of these algorithms for Dulmage-Mendelsohn decomposition and for evaluating the complexity of matroid algorithms are also derived in Section 3.3.

These algorithms are developed into a more practical version in Section 3.4. This section also considers what happens when the practical algorithm is used to process real problems that do not satisfy the generality assumption, and it is shown that the performance of the practical algorithms can be no worse than the performance of the theoretical algorithms. Finally, it reviews some implementation techniques that can greatly speed up the algorithm.

In Section 3.5 some computational results are discussed which were obtained from a test implementation of the algorithm based on the considerations of Section 3.4. While the test results are from a small sample of problems and are therefore preliminary, they are still encouraging for the eventual practical implementation of the algorithm.

Finally the current status of this research is reviewed in Section 3.6, with suggestions about some directions for future development. Particular attention is paid to possible enhancements of the algorithm that might increase its applicability.

3.2. The Matching Property and the One Row Algorithm

In this section and in Section 3.3 it is assumed that the matrix A in (3.1.3) has full rank. The effect of removing this assumption is considered in Section 3.4.

3.2.1. The Matching Property

To illustrate the pitfalls in trying to solve SP solely from the sparsity pattern of A, consider the following sparsity pattern:

$$\begin{pmatrix} \times & \times & 0 & 0 & 0 \\ 0 & \boxed{\times} & \boxed{\times} & \times & \times \\ 0 & \boxed{0} & \boxed{\times} & \times & \times \end{pmatrix}. \tag{3.2.1}$$

In order to make the first row sparser, a multiple of the second row could be added to the first to zero out the $1,2$ position. However, it appears that the $1,3$, $1,4$, and $1,5$ entries **fill-in** (change from a zero into a non-zero) because of this operation. To mitigate the fill-in, the multiple of row 3 that turns entry $1,3$ back into a zero could be added to row 1. The combination of these two row operations gives the same effect as if the boxed submatrix of (3.2.1), which is certainly non-singular, was used to turn the $1,2$ entry into a zero while keeping the $1,3$ entry zero. The expected result is

$$\begin{pmatrix} \times & 0 & 0 & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{pmatrix},$$

which is not sparser.

Consider now the following two matrices with sparsity pattern (3.2.1), transformed as above.

$$TA^1 = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & \boxed{1} & \boxed{1} & \boxed{1} \\ 0 & 0 & \boxed{1} & \boxed{1} & \boxed{1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix};$$

$$TA^2 = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & -1 & -2 \\ 0 & 1 & 1 & 2 & 3 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

In the second case, the sparsity decreased as expected, but in the first case, the sparsity *increased*. The reason for the unexpected behavior of A^1 is that the boxed submatrix has rank only 1, not 2, which caused cancellation to occur in columns 4 and 5. When a gratuitous zero appears outside the columns we were trying to affect (in the example we were trying to affect columns 2 and 3, and for A^1 gratuitous zeros appeared in columns 4 and 5), we say that **unexpected cancellation** has occurred.

Predicting unexpected cancellation can be extremely difficult. The next theorem shows that allowing unexpected cancellation makes SP essentially intractable.

Theorem 3.2.1: It is NP-Hard to solve SP.

Proof: This theorem and its proof are due to Stockmeyer (1982). It is claimed only that SP is NP-Hard, rather than NP-Complete, because it is difficult to show that SP is in NP. However, Stockmeyer has conjectured that SP $\in$ NP (see Garey and Johnson (1979) for the definition of NP).

The problem that we shall reduce to SP is

Simple Max Cut: Given an undirected graph $\mathcal{G} = (V, E)$, partition the nodes of $\mathcal{G}$ into P and $V \setminus P$ so as to maximize

$$|\{\{i,j\} \in E \mid i \in P, j \in V \setminus P\}|.$$

A proof that Simple Max Cut is NP-Complete is referenced in Garey and Johnson (1979) Problem [ND16].

Let $n = |V|$, $m = |E|$, let $A(\mathcal{G})$ be the usual $(0,1)$ vertex-edge incidence matrix of $\mathcal{G}$, and let A_i be the $n \times 2m$ matrix which is all zero except for row i, half of whose components are $+1$ and half -1. Let e be the $2m$-vector of all ones and let f be the $(2m(n+1)+1)$-vector of ones. Suppose that SP could be solved for the matrix

$$B(\mathcal{G}) = \begin{pmatrix} 0 & e & e & \cdots & e & f \\ A(\mathcal{G}) & A_1 & A_2 & \cdots & A_n & 0 \end{pmatrix}.$$

Define T^* to be a matrix so that $T^*B(\mathcal{G})$ is an optimal solution to SP. Since T^* is non-singular, it must have a perfect matching, which can be assumed without loss of generality is on its diagonal. Also, since T^* stays optimal after row scaling it can be assumed that T^* has unit diagonal. Because of the size of f, it is never worthwhile to use row 1 when reducing any other row, and hence the first column of T^* must be $(1, 0, \ldots, 0)^T$. Thus no choice for the remaining entries of the first row of T^* can cause it to be singular. Because of the column size of the A_i, and since all entries are ± 1, it is helpful to use every other row in reducing the first row. Hence, the first row of T^* must be $(1, \epsilon_1, \epsilon_2, \ldots, \epsilon_n)$, where $\epsilon_i = \pm 1$ for all $i \in V$. Let $P = \{i \mid \epsilon_i = +1\}$. Then the number of non-zeros in the first row of the reduced matrix of $B(\mathcal{G})$ is clearly

$$(2m(n+1)+1) + mn + \left(m - |\{\{i,j\} \in E \mid i \in P, j \in V \setminus P\}|\right). \tag{3.2.2}$$

But since (3.2.2) is minimized by the optimal T, P also solves the Simple Max Cut Problem for $\mathcal{G}$. □

This proof works because of the great opportunity for unexpected cancellation in a $(0,1)$-matrix. The proof shows in particular that any numerical approach to SP must be heuristic, rather than aiming for optimality. In order to "combinatorialize" SP and bring it back into the class of polynomial algorithms, an assumption about the non-zeros of A is needed that effectively rules out unexpected cancellation.

As motivation for the assumption that will be used, consider the following chain of implications about an $n \times n$ matrix B:

$$\begin{aligned} &\operatorname{rank} B = n \Leftrightarrow \det B \neq 0 \Leftrightarrow \sum_{\sigma} \operatorname{sgn}\sigma \prod_i b_{i\sigma_i} \neq 0 \\ &\Rightarrow \text{there is a permutation } \sigma \text{ such that } \quad b_{1\sigma_1}, b_{2\sigma_2}, \ldots, b_{n\sigma_n} \neq 0 \qquad (3.2.3) \\ &\Leftrightarrow B \quad \text{has a perfect matching.} \end{aligned}$$

As the example of A^1 showed, unexpected cancellation is caused by submatrices whose rank is less than that suggested by their sparsity pattern. The notion of what rank "should" be turns out to be that if a submatrix can be permuted so that it has a non-zero diagonal, then it should have full rank. Since having a non-zero diagonal is the condition in (3.2.3), rank is what it should be if the implication in (3.2.3) goes backwards as well. That is, if B has a perfect matching, then it should have full rank. Such "generality", "non-degeneracy", "general position" or "independence" is often assumed in sparse matrix studies. A formal statement of this property is:

Matching Property (MP): A has (MP) if $\operatorname{rank} A_{RC} = M(A_{RC})$ for all row subsets R and column subsets C.

In other terminology, (MP) states that term rank and numerical rank are the same for every submatrix of A. In the example above, A^2 has (MP) but A^1 does not (it fails precisely on the boxed submatrix).

It is interesting to consider how (MP) relates to other possible assumptions. A very strong assumption would be that the non-zeros of A act essentially like independent algebraic indeterminates. That is, any entry of the result of any linear algebraic operation on A can be expressed as a multivariate polynomial in formal variables x_{ij}, one for each non-zero of A. An entry of the result is considered to be zero only if its multivariate polynomial is identically zero. Algebraic independence implies (and therefore is stronger than) (MP).

A similar assumption would be that each non-zero of A is perturbed from its original value by an independent infinitesimal, similar to the construction often used to resolve degeneracy. This perturbation assumption also implies (MP). Thus (MP) is weak relative to other such assumptions.

Although (MP) is not particularly stringent, most real-life matrices do not satisfy (MP). The reason is that real matrices have many entries which are small integers, thereby producing submatrices which violate (MP). We shall nevertheless construct an algorithm to solve SP assuming (MP) because Theorem 3.2.1 implies that there is little hope of solving SP *without* such an assumption. One reasonable heuristic approach to SP is to solve it with (MP), and then to apply the resulting algorithm to matrices which do not necessarily satisfy (MP). Though it is an apparent contradiction, such an (as yet hypothetical) algorithm would be an "optimal heuristic" for SP. It would be optimal for those matrices that satisfy (MP), and it would be heuristic for the others.

3.2.2. The One Row Algorithm

For the reason discussed above, in this section and in Section 3.3 A is assumed to satisfy (MP). In order to show that (MP) implies that no unexpected cancellation can occur, some preliminary discussion is needed.

As noted in Section 3.1, solving SP involves constructing a non-singular T so that TA is as sparse as possible. By (3.2.3) T must have a perfect matching, and by permuting indices, it can be assumed that every entry on the diagonal of T is non-zero. Scaling the rows of T does not affect the sparsity of TA, and hence it can be further assumed that $t_{ii} = 1$, $i = 1, 2, \ldots, m$. With this scaling, row i of T specifies an elementary row operation to be performed on row i of A, namely add the other rows of A to the i^{th} row with the multipliers specified by the entries in row i of T. Since TA is supposed to be sparser than A, its i^{th} row should also be sparser, which leads to consideration of:

The One Row Sparsity Problem for Row i (ORSP$_i$): Find $\{\lambda_k, k \neq i\}$ so that

$$\hat{A}_{i\bullet} = A_{i\bullet} + \sum_{k \neq i} \lambda_k A_{k\bullet} \tag{3.2.4}$$

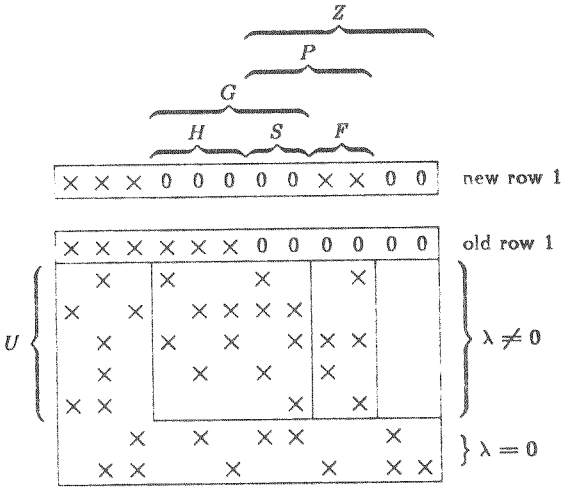
is as sparse as possible.

Once ORSP$_i$ is solved, the hope is that the resulting λ for row i of A can be packed into row i of T, which can then be used to solve SP. It is not clear that the resultant T is non-singular as required; nevertheless, in the rest of this section we shall concentrate on solving ORSP$_i$.

A set of multipliers $\{\lambda_k \mid k > 1\}$ for (3.2.4) when $i = 1$ defines the following index subsets:

$$\begin{aligned}
U &= \{k > 1 \mid \lambda_k \neq 0\},\\
H &= \{j \mid \hat{a}_{1j} = 0 \text{ and } a_{1j} \neq 0\},\\
S &= \{j \mid \hat{a}_{1j} = 0 \text{ and } a_{1j} = 0 \text{ and } a_{kj} \neq 0 \text{ for some } k \in U\},\\
G &= H \cup S,\\
F &= \{j \mid \hat{a}_{1j} \neq 0 \text{ and } a_{1j} = 0\},\\
P &= F \cup S = \{j \mid a_{1j} = 0 \text{ and } a_{kj} \neq 0 \text{ for some } k \in U\}, \text{ and}\\
Z &= \{j \mid a_{1j} = 0\}.
\end{aligned}$$

That is, U is the set of **used** rows; H is the set of **hit** columns, where a non-zero was changed to a zero; S is the set of **saved** columns, where a zero that might have been expected to be filled-in (since $a_{kj} \neq 0$) was not filled-in; G is the set of **good** columns, where the entry was actively manipulated for the better; F is the set of **filled-in** columns; P is the set of **potential fill-in** columns; and Z is the set of **zero** columns. As an example of these definitions, the index sets are indicated for the following sparsity pattern:



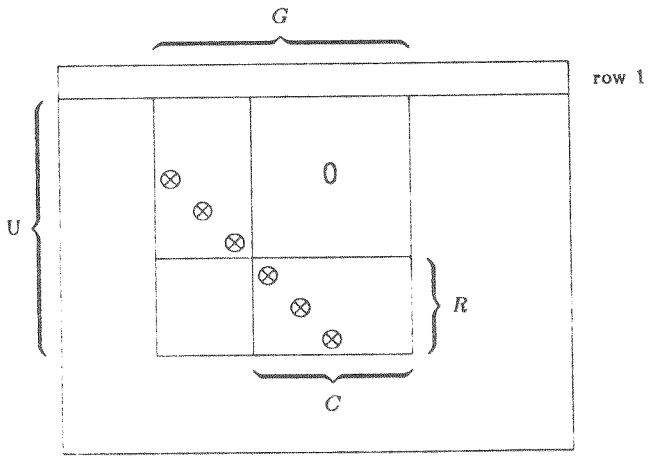
Note that the net decrease in non-zeros in row 1 is $|H| - |F|$, so that solving ORSP_1 is equivalent to solving $\max_\lambda |H| - |F|$.

Now (MP) can be put to work. The next theorem states the intuitive result that if k columns of row 1 are affected for the good, at least k rows must be used. This theorem is the key fact that rules out unexpected cancellation.

Theorem 3.2.2: For any λ, $|G| \leq |U|$.

Proof: By contradiction. Assume that $|G| > |U|$.

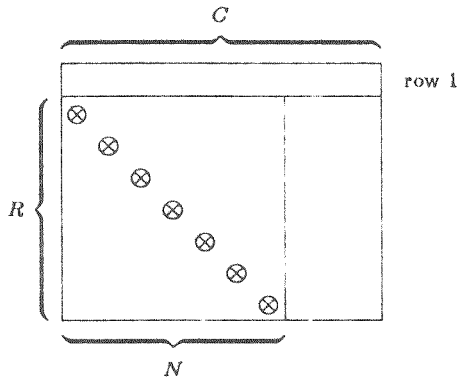
If $M(A_{UG}) < |U|$ then by Proposition 3.1.1 A must look like



(3.2.5)

where the row subset R and column subset C are defined as shown. (When we say that a matrix "looks like" a picture such as the one above, we mean that its rows and columns can be permuted so that it has the form shown.) Otherwise ($M(A_{UG}) = |U|$), let $R = U$ and $C = G$. In either case, A has a submatrix

that looks like



(3.2.6)

where $R \subseteq U$, $C \subseteq G$ and $|R| < |C|$. Also, the rows of R provide the only non-zero contribution to the new row 1 in the columns of C, and $M(A_{RC}) = |R|$. The set of columns induced by a maximum matching in A_{RC} is denoted by N as shown in (3.2.6).

Note that A_{RN} is square and $M(A_{RN}) = |R|$; thus, by (MP), A_{RN} is non-singular. Now $A_{1N} + \lambda_R^T A_{RN} = \hat{A}_{1N} = 0$ since $N \subseteq G$, so that λ_R must be the unique solution of $\lambda_R^T A_{RN} = -A_{1N}$. Define $\tilde{R} = R \cup \{1\}$. For $k \in R$ let $R_k = \tilde{R} \setminus \{k\}$, and for $j \in C \setminus N$ let $N_j = N \cup \{j\}$. Then Cramer's Rule implies that

$$\lambda_k = (-1)^k \det A_{R_k N} \quad \text{for } k \in R. \tag{3.2.7}$$

For $j \in C \setminus N$ (which is non-empty), using (3.2.7) and the expansion of the determinant by cofactors in reverse, it can be shown that

$$\hat{a}_{1j} = \det A_{\tilde{R}N_j}. \tag{3.2.8}$$

Note that $j \in C \subseteq G$ implies that $\hat{a}_{1j} = 0$. If $a_{1j} \neq 0$, then the perfect matching in A_{RN} can be trivially extended by a_{1j} to a perfect matching in $A_{\tilde{R}N_j}$. But now (MP) implies by (3.2.8) that $\hat{a}_{1j} \neq 0$, a contradiction.

Suppose instead that $a_{1j} = 0$. Then $j \in S$, and therefore there must be some $k \in R$ such that $a_{kj} \neq 0$. Since $k \in R \subseteq U$, it follows that $\lambda_k \neq 0$. Thus, by (3.2.6), $A_{R_k N}$ must have a perfect matching. But adding a_{kj} to this matching gives a perfect matching of $A_{\tilde{R}N_j}$. Once again, (MP) and (3.2.8) imply that $\hat{a}_{1j} \neq 0$, a contradiction. □

Corollary 3.2.3: $M(A_{UG}) = |G|$; thus, by (MP), $\operatorname{rank} A_{UG} = |G|$.

Proof: If $M(A_{UG}) < |G|$, then by Proposition 3.1.1 A must look like (3.2.5), which would again lead to a submatrix of A like (3.2.6), which cannot exist by Theorem 3.2.2. □

Theorem 3.2.2 and its corollary prove that there can be no unexpected cancellation when (MP) is satisfied. Indeed, (3.2.6) is precisely a picture of what is meant by unexpected cancellation.

It is possible to have $|U| > |G|$, but it causes additional work with no further increase in sparsity. A subset $R \subseteq U$ could be selected that perfectly matches into G (which is possible by Corollary 3.2.3), and the non-zero part of λ computed as the solution of

$$\lambda_R^T A_{RG} = -A_{1G}, \tag{3.2.9}$$

which gives the same result with less work. Conversely, suppose that a square, non-singular submatrix A_{RG} is used to to zero out A_{1G} via (3.2.9). Then Theorem 3.2.2 implies that no columns outside G are affected for the good (either hit or saved).

Hence it can be assumed without loss of generality that $|U| = |G|$. Since λ is now uniquely determined by an equation like (3.2.9), U and G can be thought of as determining λ rather than vice-versa. Thus ORSP_i has been reduced to the more combinatorial problem of finding optimal U and G.

Since finding λ requires solving a system of linear equations of size $|U|$, if there are several different optimal U the smallest cardinality one is preferred in order to minimize work. (Actually, as was mentioned in Section 3.1, since work is proportional to the number of non-zeros, the number of non-zeros in A_{UG} should be minimized, but $|U|$ makes an acceptable substitute.) Formally stated, we would ideally like to solve:

The Strong ORSP$_i$: Find an optimal solution to ORSP$_i$ that minimizes $|U|$.

A little reflection over the definitions of the index sets F, S, H and P reveals that they can be easily determined combinatorially from U and G. In fact, P depends only on U (in words, P is the set of not-identically zero columns j of $A_{U\bullet}$ such that $a_{1j} = 0$), and so it will be denoted by $P(U)$. The set G merely determines how $P(U)$ is split up into F and S. These observations lead to an easy proof that the gain of zeros, $|H| - |F|$, depends only on U, not on G:

Theorem 3.2.4: Let G_1 and G_2 be two sets of columns that perfectly match into U, and denote the set of hit columns corresponding to G_i by H_i, $i = 1, 2$, etc. Then

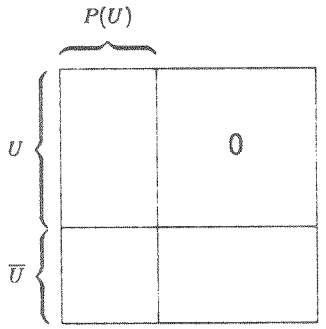
$$|H_1| - |F_1| = |H_2| - |F_2|.$$

Proof: It is easy to see that $|H_i| = |U| - |S_i|$ and $|F_i| = |P(U)| - |S_i|$, so that $|H_i| - |F_i| = |U| - |P(U)|$, $i = 1, 2$. □

The proof shows that solving ORSP$_i$ is equivalent to solving

$$\max_U |U| - |P(U)|. \tag{3.2.10}$$

Since A has full rank, every U must perfectly match into some column subset G; thus, the maximization in (3.2.10) is over all U. Define $R = \{2, 3, \ldots, m\}$ and $\overline{U} = R \setminus U$, and call A_{RZ} the **first zero-section of** A. By the definition of $P(U)$, A_{RZ} must look like



Thus every non-zero in A_{RZ} is contained in either a column of $P(U)$ or a row of $\overline{U}$. Consider R and Z to be disjoint sets whose union is the set of **lines** of A_{RZ}. Then, since every non-zero of A_{RZ} is in a line of $P(U) \cup \overline{U}$, A_{RZ} is said to be **covered** by the lines in $P(U) \cup \overline{U}$. Conversely, suppose that L is a cover of the first zero-section by lines with a minimal number of columns. Then define $U = R \setminus \{\text{rows in } L\}$. Since L has a minimal number of columns, the columns in L must be $P(U)$ (otherwise a column could be dropped from L and it would still cover A_{RZ}, contradicting minimality). Hence, covers by lines with minimal columns correspond to subsets $U \subseteq R$.

But now ORSP$_1$ has been reduced to

$$\begin{aligned} \max_U |U| - |P(U)| &= (m-1) - \min_U \big(|P(U)| + |\overline{U}|\big) \\ &= (m-1) - \min_{L \text{ covers } A_{RZ}} |L|, \end{aligned} \tag{3.2.11}$$

so that finding a minimum cardinality cover of the first zero-section also solves ORSP$_1$.

The classic combinatorial duality theorem of König and Egervary (Ryser (1963), Theorem 5.1) shows that a minimum cover in (3.2.11) can be computed via a maximum matching in A_{RZ}:

Theorem 3.2.5: $M(A_{RZ}) = \min\{\, |L| \mid L \text{ covers } A_{RZ} \,\}$, and maximum matchings and minimum covers are dual combinatorial objects (which means that any algorithm that computes one must also compute the other, at least implicitly). □

Finding maximum matchings in bipartite graphs is a well-studied problem, for which many polynomial algorithms have been developed. Most such algorithms find a maximum matching by labelling. If labels are started at the row side of the bipartite graph, then at the final iteration of a labelling algorithm the labelled rows and columns are those reachable by a partial augmenting path from an unmatched row. The dual minimum cover can be extracted from the final labels as $\{$ labelled columns $\} \cup \{$ unlabelled rows $\}$ (see Lawler (1976), p. 193). Since U is complementary to the rows in a minimum cover, U is the set of labelled rows at the end of a maximum matching algorithm (and $P(U)$ is the set of labelled columns). In matrix terms, if the first zero-section is the matrix in (3.1.5), then U is the set of rows of C. This gives a polynomial algorithm for solving ORSP$_i$.

There is still more juice that can be squeezed out of the maximum matching. Recall that it is preferable to solve the Strong ORSP$_i$, *i.e.*, find the minimum cardinality U among all optimum U. Note that it is easy to turn the maximum matching problem on A_{RZ} into a network flow problem (see Section 2.5). In the network flow context, the dual object to a maximum flow is a minimum cut. At an optimum of a network flow problem, a minimum cut can be extracted as the set of vertices reachable from the source by a partial augmenting path (as above for minimum covers). Let the minimum cut computed in this way be called the **standard** minimum cut. Since labels for maximum flow and maximum matching are isomorphic, the standard minimum cut (standard minimum cover in matrix terms), must be equal to $\overline{U} \cup P(U)$.

Theorem 3.2.6: In any network the standard minimum cut is a subset of every minimum cut. Thus the standard minimum cut is the same for every optimal flow. Hence its definition is independent of the optimal flow used to compute it, and it has minimum cardinality among all minimum cuts (see Ford and Fulkerson (1962), p. 13). □

Since the standard minimum cut and the standard minimum cover have complementary sets of rows, U is the set of rows in the standard minimum cut. Such a U is already known to solve ORSP$_i$ by (3.2.11). Theorem 3.2.6 states that the standard U is unique, solves the Strong ORSP$_i$, and can be found at no additional expense. Returning to the partition (3.1.5), if A_{RZ} has several such partitions in which C has different sizes, then by Theorem 3.2.6, maximum matching automatically generates the partition in which C has the fewest rows possible. In addition to cutting down on the amount of work needed to solve equations for λ in practice, this theorem has important theoretical implications that are explored in subsequent sections.

Note that there is a strong asymmetry in choosing which side of the network is the source. The above discussion applies when the maximum matching or network flow is started from the *row* side. If it is started from the column side, then the *largest* U that is optimal for ORSP$_i$ is generated instead of the smallest.

We shall now put all the pieces together into an algorithm for solving (the Strong) ORSP$_i$.

The One Row Algorithm for Row i (ORA$_i$):

0. Perform a maximum matching in the i^{th} zero-section (starting from the row side).
1. Construct U_i as the set of labelled rows at the end of the matching.
2. Find a column subset G_i that perfectly matches into U_i (then $A_{U_iG_i}$ is non-singular by (MP)).
3. Solve $\lambda_{U_i}^T A_{U_iG_i} = -A_{iG_i}$.
4. Set $\hat{A}_{i\bullet}$ to $A_{i\bullet} + \lambda_{U_i}^T A_{U_i\bullet}$.

Note that Step 2 allows considerable freedom in choosing G_i, a point whose importance is shown later.

3.3. Theoretical Algorithms for SP

We would like to combine the local solutions to ORSP$_i$, $i = 1, 2, \ldots, m$ into a global solution for SP. However, this process requires care. For example, consider the matrix

$$A = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}. \tag{3.3.1}$$

Denote the (unique by Theorem 3.2.6) optimal U for ORSP$_i$ by U_i^*. Then for (3.3.1), $U_1^* = \{2\}$ and $U_2^* = \{1\}$. (When the i^{th} zero-section has no columns as in this case, the bipartite graph of the zero section has only left (row) vertices and no edges. Hence a matching algorithm terminates after it has initially

labelled all of the rows. Since U_i^* is the set of labelled rows at optimality, when row i is completely dense $U_i^* = \{1, 2, \ldots, m\} \setminus \{i\}$.) At row 1 G_1 can be chosen to be $\{1\}$, giving $\hat{A}_{1\bullet} = (0 \quad -1)$. In row 2, G_2 can also be chosen to be $\{1\}$, giving $\hat{A}_{2\bullet} = (0 \quad 1)$. With these choices the final result is

$$\hat{A} = \begin{pmatrix} 0 & -1 \\ 0 & 1 \end{pmatrix}$$

which is singular. This illustrates that the G_i cannot be chosen arbitrarily in Step 2 of ORA$_i$.

3.3.1. The Parallel Algorithm

There is a fairly natural way to choose G_i in Step 2 that avoids the potential difficulty above, and that also saves time. Let M be a fixed maximum matching for A (M must be row-perfect since A has full row rank). Once U_i^* is determined in Step 1 of ORA$_i$, G_i can be chosen to be the set of columns that U_i^* matches to under M. Note that $\hat{A}$, the output of running ORA$_i$ for $i = 1, 2, \ldots, m$, equals TA, where the i^{th} row of T has one in the diagonal position, and is the λ from running ORA$_i$ elsewhere. Define T^* to be the T obtained by choosing the G_i relative to M. Thus when $i \neq j$, $t_{ij}^* \neq 0$ if and only if $j \in U_i^*$.

Suppose that it could be shown that T^* is non-singular, so that it is a valid candidate for a T to solve SP. Then T^* must be an optimal solution to SP. Consider any other candidate T, which can be assumed without loss of generality to have a diagonal of all ones (as in Section 3.2). If T increased the sparsity of A more than T^*, then at least one row of TA would have to be sparser than the corresponding row of T^*A. But the optimality of ORA$_i$ implies that every row of T^*A is individually as sparse as possible. Thus T^* is optimal if it is non-singular.

The determination of the non-singularity of T^* depends on the implications of the uniqueness of the U_i^* for the structure of T^*. Define a directed graph D^* with nodes $\{1, 2, \ldots, m\}$ and arcs $\{(k, i) \mid k \in U_i^*\}$, so that D^* captures the sparsity pattern of T^*. Such a directed graph can be similarly defined for any square sparse matrix T with non-zero diagonal.

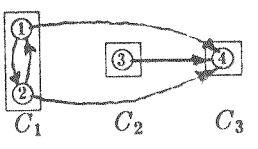
In any directed graph D, node j is defined to be **reachable** from node i if there is a directed path from i to j. The relation

$$i \sim j \Leftrightarrow i \text{ is reachable from } j \textit{ and } j \text{ is reachable from } i$$

is an equivalence relation that partitions the nodes of D. A class of this partition is called a **strong component** of D: every node in a strong component C is reachable from every other node in C. If a node l in strong component C_j is reachable from a node k in strong component C_i, then C_i **precedes** C_j, which is denoted $C_i \prec C_j$. It can be shown that the $\prec$ relation is a partial order on the strong components.

To reflect the strong component partition back into matrix terms, order the strong components of D in a linear order consistent with $\prec$, and order nodes arbitrarily within components. If the corresponding principal permutation is applied to T, it induces a block lower triangular structure, where the diagonal blocks are irreducible and correspond to the strong components. This decomposition of T is essentially unique and is called the (square) **Dulmage-Mendelsohn decomposition** (or DM decomposition) of T. The DM decomposition is studied more closely at the end of this section. See, *e.g.*, George and Gustavson (1980) for details of this decomposition. For example, if

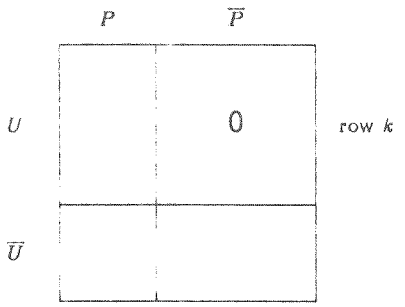
$$T = \begin{array}{c} \begin{array}{cccc} C_1 & C_2 & C_3 & \end{array} \\ \left(\begin{array}{ccc|c} \times & \times & & \\ \times & \times & & \\ \hline & & \times & \\ \hline \times & \times & \times & \times \end{array}\right) \end{array}, \quad \text{then } D = \begin{array}{c} \\ \\ \end{array}$$



where the decompositions of T and D are indicated by boxes. The next theorem shows that the block triangular decomposition of T^* has a very special structure.

Theorem 3.3.1: If $l \neq i$, $l \in U_k^*$ and $k \in U_i^*$, then $l \in U_i^*$.

Proof: For ease of notation, let $U = U_i^*$, $\overline{U} = \{1, 2, \ldots, m\} \setminus \{i\} \setminus U_i^*$, $P = P(U_i^*)$ and $\overline{P} = Z_i \setminus P(U_i^*)$. Thus U and $\overline{U}$ partition the rows of the i^{th} zero-section, and P and $\overline{P}$ partition its columns. By definition of $P(U)$ and k, the i^{th} zero-section looks like



Since U_i^* corresponds to a maximum matching, by Proposition 3.1.1 $A_{\overline{U}P}$ has a row-perfect matching, and so at least $|\overline{U}|$ lines are necessary to cover it. Note that since $A_{k\overline{P}} = 0$ and $k \notin \overline{U}$, $A_{\overline{U}\overline{P}}$ is a submatrix of the k^{th} zero-section.

Let L_k^* be the standard minimum cover of the k^{th} zero-section by lines. Recall that the standard minimum cover has the largest number of rows among all minimum covers. Consider the set of lines $L_k = L_k^* \cup \overline{U} \setminus \overline{P}$. The set L_k does not contain the columns of $\overline{P}$, and so it might not cover $A_{*\overline{P}}$, which is part of the k^{th} zero-section. Since the only difference between L_k and L_k^* is in lines passing through $A_{*\overline{P}}$, L_k does cover the rest of the k^{th} zero-section. But the only non-zero rows of $A_{*\overline{P}}$ are the rows in $\overline{U}$, and since $\overline{U} \subseteq L_k$, L_k also covers the k^{th} zero-section.

L_k^* and L_k have the same set of lines outside $A_{\overline{U}P}$. Since $A_{\overline{U}P}$ has a row-perfect matching, the number of lines of L_k^* passing through $A_{\overline{U}P}$ must be at least $|\overline{U}|$. The lines in L_k passing through $A_{\overline{U}P}$ are precisely $\overline{U}$, the minimum possible number, and so overall L_k contains at most as many lines as L_k^*. But since L_k^* is minimum, L_k must also be a minimum cover

Finally, note that L_k contains at least as many rows as L_k^* does. But since L_k^* is the (unique) standard minimum cover, L_k^* has the maximum possible number of rows among all minimum covers. Thus L_k must equal L_k^*, so that $\overline{U} \subseteq \{\text{rows of } L_k^*\}$. Taking complements gives $U_i^* \cup \{i\} \supseteq U_k^* \cup \{k\}$, which gives $U_k^* \setminus \{i\} \subseteq U_i^*$, as desired. □

In graph terms, Theorem 3.3.1 implies that if l, k and i are distinct nodes of $\mathcal{D}^*$, and (l, k) and (k, i) are edges of $\mathcal{D}^*$, then (l, i) must also be an edge of $\mathcal{D}^*$. When this is true of an a directed graph, it is said to be **transitively closed**. Applying the theorem inductively, j is reachable from i in $\mathcal{D}^*$ if and only if (i, j) is an edge of $\mathcal{D}^*$. In terms of the block lower triangular structure of T^*, this result implies that the diagonal blocks of T^* are all completely dense, and the subdiagonal blocks are either all zero or completely dense. In particular, if i and j are in the same strong component C of $\mathcal{D}^*$, rows i and row j of T^* have the same sparsity pattern, so that $C \subseteq U_i^* \cup \{i\} = U_j^* \cup \{j\}$. These insights into the structure of T^* are crucial to the proof of the next theorem.

Theorem 3.3.2: T^* is non-singular.

Proof: It can be assumed without loss of generality that M matches row i to column i in A, $i = 1, 2, \ldots, m$, and that the indices of A and T are ordered so that T^* is block lower triangular. Since T^* is block lower triangular, it suffices to show that each diagonal block of T^* is non-singular.

Recall that the effect of ORA_i is to make $\hat{A}_{iG_i} = 0$. By the choice of M, and G_i relative to M, $G_i = U_i^*$, so that $\hat{A}_{iU_i^*} = 0$. Since $i \notin U_i^*$, the blocks of $\hat{A}$ corresponding to the diagonal blocks of T^* are diagonal matrices; the subdiagonal blocks of $\hat{A}$ corresponding to dense subdiagonal blocks of T^* are completely zero. Thus a typical A and T^* might look like

$$\underset{T^*}{\begin{pmatrix} F & 0 & 0 \\ 0 & F & 0 \\ 0 & F & F \end{pmatrix}} \cdot \underset{A}{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}} = \underset{\hat{A}}{\begin{pmatrix} D & * & * & * \\ * & D & * & * \\ * & 0 & D & * \end{pmatrix}},$$

where "F" represents a full or dense submatrix, "D" represents a diagonal submatrix, "0" represents a zero submatrix, and "*" represents an arbitrary submatrix.

Denote the set of indices of a typical diagonal block of T^* by C. For $i \in C$, let $U_C = U_i^* \cup \{i\}$; as discussed above, this definition is independent of i, and $C \subseteq U_C$. If $C = U_C$, as is the case with the first two blocks in the example, then $T^*_{CC} A_{CC} = \hat{A}_{CC}$, which is diagonal, and which has non-zero diagonal since there can be no unexpected cancellation. Since M restricted to A_{CC} is a perfect matching for A_{CC}, A_{CC} is non-singular by (MP). But then $T^*_{CC} = \hat{A}_{CC} A^{-1}_{CC}$, and so T^*_{CC} is non-singular.

Suppose instead that $C \subset U_C$, as in the third block of the example, and define $L = U_C \setminus C$. Thus

$$A_{U_C U_C} = \begin{pmatrix} A_{LL} & A_{LC} \\ A_{CL} & A_{CC} \end{pmatrix}.$$

The submatrix T^*_{CC} satisfies the following equations by definition of T^*, C and L:

$$\begin{aligned} T^*_{CL} A_{LL} + T^*_{CC} A_{CL} &= 0 \\ T^*_{CL} A_{LC} + T^*_{CC} A_{CC} &= \hat{A}_{CC}, \quad \text{which is diagonal.} \end{aligned} \tag{3.3.2}$$

Since M restricted to $A_{U_C U_C}$ is a perfect matching for both $A_{U_C U_C}$ and A_{LL}, by (MP) both $A_{U_C U_C}$ and A_{LL} are non-singular. Since A_{LL} is non-singular, T^*_{CC} can be partially solved for in (3.3.2) to get

$$T^*_{CC}\left(A_{CC} - A_{CL} A^{-1}_{LL} A_{LC}\right) = \hat{A}_{CC}. \tag{3.3.3}$$

The matrix $A_{CC} - A_{CL} A^{-1}_{LL} A_{LC}$ in (3.3.3) is called the **Schur complement** of A_{LL} in $A_{U_C U_C}$, denoted $(A_{U_C U_C}/A_{LL})$. It is well known (see Cottle (1974), equations (2) and (4)) that when $A_{U_C U_C}$ is non-singular, $(A_{U_C U_C}/A_{LL})$ is non-singular if and only if A_{LL} is non-singular. Since A_{LL} is non-singular here, T^*_{CC} can be fully solved for in (3.3.3) as the product of two non-singular matrices, implying that T^*_{CC} is non-singular in this case as well. □

Since T^* is non-singular, it can be used to transform A into $\hat{A}$. Generating $\hat{A}$ via T^* processes each row in parallel, *i.e.* each row is solved relative to the original matrix rather than relative to a partially transformed matrix. We call this procedure the

Parallel Algorithm (PA):

Find a maximum matching M of A.
For $i = 1, 2, \ldots, m$
  Generate row i of $\hat{A}$ from A using ORA_i, picking G_i relative to M.
  End.

The results in this section make it easy to prove the next theorem.

Theorem 3.3.3: PA solves SP.

Proof: T^* is non-singular by Theorem 3.3.2. Since every row of A is made as sparse as possible in $\hat{A}$, T^* must be optimal. □

3.3.2. The Sequential Algorithm

The parallelism of PA seems unsatisfactory for three reasons. First, it is more natural to process A **sequentially**, *i.e.* by solving each row's matching problem on the partially reduced A whose previous rows have already been processed. Second, by processing A sequentially, $\hat{A}$ can be overwritten on A, thereby saving space. Also, it is shown later that the optimal U's can only become smaller, which saves time in solving equations (3.2.9). Third, and most important, with PA the flexibility in choosing G_i at Step 2 of ORA_i is lost. Having flexibility in choosing G_i is important when the algorithm is applied to real problems that might not satisfy (MP). It is important because in practice $A_{U_i^* G_i}$ might be singular despite having a perfect matching.

For these reasons, we consider the following algorithm, which overcomes the objections to PA:

Sequential Algorithm (SA):

For $i = 1, 2, \ldots, m$

Apply ORA_i to A, choosing G_i in any way desired, as long as $A_{U_i^* G_i}$ is non-singular.

Replace row i of A by the generated row i of $\hat{A}$.

End.

Note that A is dynamically changed every time through the loop in SA, so that (MP) might no longer hold for the partially transformed A. However, we know of no example where (MP) fails. By the nature of ORA_i, each iteration of the loop in SA is equivalent to left-multiplication of A by an elementary transformation. That is, at iteration i of the loop, the current A is left-multiplied by matrix E^i, where E^i is the identity except that row i of E^i is the λ from ORA_i. Since all the E^i are non-singular, the output matrix of SA is equivalent to the original A.

It was easy to prove that PA produces an optimally sparse answer, but difficult to prove that its output matrix is equivalent to the input. For SA, the situation is just the reverse. The next theorem uses the optimality of PA to show that SA is optimal.

Theorem 3.3.4: SA produces the same final number of non-zeros as PA, and hence SA also solves SP.

Proof: The U computed for row i by ORA_i in PA is still denoted by U_i^*, and the (possibly different) U computed for row i in SA is denoted by U_i. Denote by A^i the partially reduced A at the beginning of the i^{th} iteration of SA (just before replacing row i), so that the original input A equals A^1. The proof is by induction on the row index i; we shall prove row by row that SA produces the same reduction in non-zeros as PA. The hypothesis $U_k \subseteq U_k^*$ for all $k < i$ is also carried through the induction. At row 1, $U_1^* = U_1$ since $A = A^1$, and SA and PA both reduce row 1 by the same amount, proving the base of the induction.

For row i, let $R = \{1, 2, \ldots, m\} \setminus \{i\}$ and Z be the row and column indices of the i^{th} zero-section for A^i. Since row i has not yet been changed in A^i, R and Z are also the index sets of the i^{th} zero-section for A^1. Recall that $L^* = R \setminus U_i^* \cup P(U_i^*)$ is the standard minimum cover of A^1_{RZ} by lines. The first claim is that L^* is also a covering of A^i_{RZ} by lines. Let $\overline{P} = Z \setminus P(U_i^*)$. The only way for L^* *not* to be a cover is if the result of a previous operation has introduced a non-zero into $A^i_{U_i^* \overline{P}}$ (the part of the i^{th} zero-section not covered by L^*).

Suppose that the first non-zero introduced into $A^i_{U_i^* \overline{P}}$ is in row l of U_i^*, and that it originated from row k while processing row $l < i$, so that $k \in U_l$. By the induction hypothesis, $U_l \subseteq U_l^*$, implying that $k \in U_l^*$. Since $l \in U_i^*$ and $k \in U_l^*$, and since k clearly cannot equal i, by Theorem 3.3.1 $k \in U_i^*$. Since iteration l is the *first* one where $A^i_{U_i^* \overline{P}}$ became non-zero, and since $k \in U_i^*$, $A^l_{k\overline{P}}$ is zero. But then row k cannot introduce a non-zero into $A^i_{U_i^* \overline{P}}$. Thus $A^i_{U_i^* \overline{P}}$ must still be zero, and so L^* is a cover for A^i_{RZ}.

$$
\begin{aligned}
\text{Now } |L^*| &= M(A^1_{RZ}) && \text{(by Theorem 3.2.5)} \\
&= \text{rank } A^1_{RZ} && \text{(since (MP) } \textit{does} \text{ hold for } A^1\text{)} \\
&= \text{rank } A^i_{RZ} && \text{(since } A^i_{RZ} \text{ is a non-singular transformation of } A^1_{RZ}\text{)} \\
&\leq M(A^i_{RZ}) && \text{(by (3.2.3))} \\
&\leq |L^*| && \text{(since any cover provides an upper bound for } M(A^i_{RZ})\text{.)}
\end{aligned}
$$

Thus $|L^*| = M(A^i_{RZ})$, and so by Theorem 3.2.5, L^* must be a minimum cover for A^i_{RZ}. However, L^* might not be the minimum cover whose complementary U is smallest. Since U_i^* is the complementary U for L^*, and U_i is the complementary U for the standard minimum cover for A^i_{RZ}, Theorem 3.2.6 ensures that $U_i \subseteq U_i^*$. This verifies one of the induction hypotheses.

Recall from (3.2.11) that the reduction in non-zeros from solving ORSP_i is $(m-1) - |L|$, where L is a minimum cover for the i^{th} zero-section. The chain of equalities above shows that $M(A^1_{RZ}) = M(A^i_{RZ})$. By Theorem 3.2.5, the minimum covers for A^1_{RZ} and A^i_{RZ} then have the same size, and the reduction in non-zeros is the same for both. But the reduction in non-zeros for A^i_{RZ} is achieved by PA, which is optimal for row i, and so SA must also be optimal for row i. □

The proof of Theorem 3.3.4 produces the bonus that the sequential U's are (if anything) *smaller* than the parallel U's, so that SA needs to solve smaller linear systems to obtain λ.

3.3.3. The SP Decomposition

The proof of Theorem 3.2.2 shows that the U_i^* are strongly interrelated, and that their joint structure is related to the block triangular decomposition. Block triangular decomposition was developed by Dulmage and Mendelsohn (1963) for rectangular matrices, but our interest is in DM decomposition of square matrices, as discussed in the remarks before Theorem 3.2.2. The theory of decompositions has recently been considerably generalized by Iri (1983).

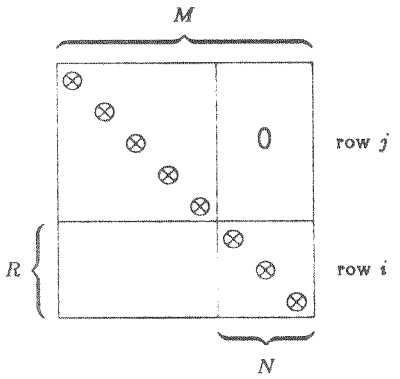
Let M be a row-perfect matching in A, and let $m(i)$ be the column to which row i matches under M. Define the directed graph D_M to have nodes $\{1, 2, \ldots, m\}$ and arcs $\{(i,j) \mid a_{j,m(i)} \neq 0\}$. We say that row j is reachable from row i via M, denoted $i \underset{M}{\rightarrow} j$, if j is reachable from i in D_M. For example, if

$$A = \begin{pmatrix} \otimes & 0 & 0 \\ \times & \otimes & \times \\ 0 & \times & \otimes \end{pmatrix},$$

and M is the circled matching, then $1 \underset{M}{\rightarrow} 3$ but $3 \underset{M}{\not\rightarrow} 1$. Note that this concept essentially depends only on the square submatrix of A induced by the matched columns of M. The next theorem illuminates the connection between the U_i^* and DM decomposition.

Theorem 3.3.5: $i \in U_j^*$ if and only if $i \underset{M}{\rightarrow} j$ for all row-perfect matchings M of A.

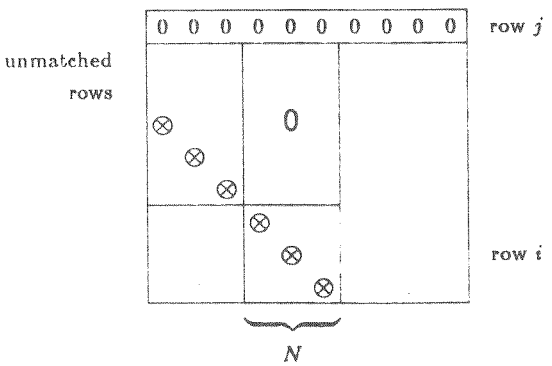
Proof: Two facts are needed for this proof. Fact 1 is that any (partial) matching M can be extended to a maximum matching $\overline{M}$ that uses the same columns as M (see Lawler (1976), Theorem 5.4.1). Fact 2 is that $i \underset{M}{\not\rightarrow} J$ if and only if the square submatrix of A induced by the columns of M can be partitioned like



(3.3.4)

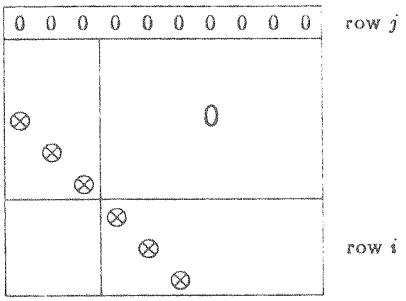
where R is the subset of rows k for which $i \underset{M}{\rightarrow} k$, M is the subset of columns matched under M, and $N = \{j \mid m(k) = j \text{ for some } k \in R\}$.

(Proof of $\Rightarrow$) Assume that $i \in U_j^*$, but that there is a row-perfect matching M such that $i \underset{M}{\not\rightarrow} j$, and so by Fact 2, $A_{\bullet M}$ looks like (3.3.4). Let N be M restricted to $A_{\bullet N}$, and note that $A_{\bullet N}$ is part of the j^{th} zero-section. Using Fact 1, extend N to a maximum matching in the j^{th} zero-section, which then looks like

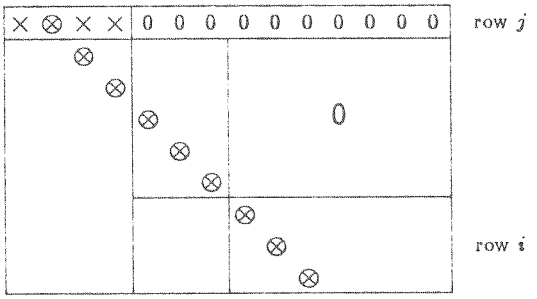


Because of the zeros above the matched part of A_{*N}, there is no way to label a row of R starting at an unmatched row. But then $i \notin U_j^*$, a contradiction.

(Proof of $\Leftarrow$) Assume that $i \underset{M}{\rightarrow} j$ for all row-perfect M, but that $i \notin U_j^*$. Since $i \notin U_j^*$, i is not in the standard minimum cover of the j^{th} zero-section. Hence, the j^{th} zero-section with a maximum matching N in it must look like



By Fact 1, N can be extended to a row-perfect $\overline{N}$ in all of A that uses the same columns as N, so that A looks like



But now $i \underset{N}{\not\rightarrow} j$ by Fact 2, a contradiction. □

The remarks after Theorem 3.3.1 show that the U_i^* induce a decomposition, which we call the **SP decomposition**, on the rows of A. That is,

$$i \sim j \Leftrightarrow i \in U_j^* \text{ and } j \in U_i^*$$

is an equivalence relation which induces an ordered partition on the rows of A. Stated another way, $i \sim j$ if and only if i and j belong to the same diagonal block of T^*. Because of the uniqueness of the U_i^*, the SP decomposition is an intrinsic or canonical property of a sparse matrix. Theorem 3.3.5 was motivated by curiosity about the relationship of the SP decomposition to the DM decomposition, which is also canonical. The content of Theorem 3.3.5 is that the SP decomposition is the coarsest partition of the rows that is a refinement of every *square* DM decomposition.

3.3.4. The Complexity of the Null Space Problem

Another interesting theoretical consequence of PA and SA applies to computational complexity. Our research into SP was originally motivated by a different problem, the

Null Space Problem (NSP): Let $m < n$. Given a sparse $m \times n$ full-rank matrix A, find an $(n-m) \times n$ **null space** matrix Z of full rank so that (1) $ZA^T = 0$ and (2) Z is as sparse as possible.

NSP is also an important problem to solve for large-scale constrained optimization, since so-called **null space** methods need such a Z to operate efficiently (see Gill, Murray and Wright (1981), Section 5.1.3; see also Kaneko, Lawo and Thierauf (1982) for a heuristic approach to NSP for a subclass of matrices).

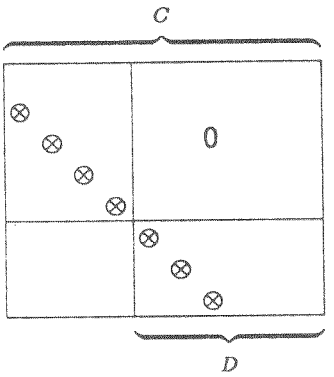
Suppose that there were a polynomial algorithm for solving NSP (say, Algorithm Z). Apply Algorithm Z to A to obtain an optimal Z^*. Now apply Algorithm Z to Z^* to obtain Z^{**}. Since A spans the null space of Z^*, by simple linear algebra Z^{**} must be equivalent to A. If there were any other matrix equivalent to

A and sparser than Z^{**}, then Z^{**} would not be an optimal null space basis for Z^*. Thus Z^{**} must be the sparsest possible equivalent matrix to A, so that Z^{**} solves SP for A. In a vague sense then, SP = (NSP)^2.

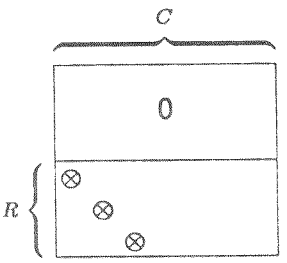
It is fascinating that solving NSP is NP-Complete, even with a stronger generality assumption about A than (MP). Some preliminary discussion is needed to show this result.

Suppose that there is a vector z^* in the null space of A that has fewer non-zero entries than any of the rows of Z^*. If z^* is adjoined to Z^*, a dependency that includes z^* is created among the rows of the augmented matrix. If a row of Z^* which is dependent on z^* in the augmented Z^* is deleted, the resulting matrix still spans the null space of A, and is even sparser than Z^*. Therefore an optimal Z^* must include a sparsest possible null-space vector.

Now consider the subset C of columns of A picked out by the non-zero entries of such a z^*. Since $A_{\bullet C} z^*_C = 0$, the columns in C must be dependent. By (MP) or any stronger assumption, since $A_{\bullet C}$ does not have full column rank, it cannot have a column-perfect matching. By Proposition 3.1.1 $A_{\bullet C}$ looks like



with column subset D defined as indicated. The columns of D must also be dependent, but if $|D| < |C|$, there is a sparser null-space vector than z^* based on D, contradicting that z^* is sparsest possible. Thus $A_{\bullet C}$ must look like



where R is the set of non-zero rows in $A_{\bullet C}$. If $|R| + 1 < |C|$, a column of C could be dropped and the columns in C would still be dependent, so that $|R| + 1 = |C|$. For $j \in C$, let $C_j = C \setminus \{j\}$. Then $M(A_{RC_j})$ must equal $|R|$ for all $j \in C$, for otherwise the size of C could be diminished, again contradicting optimality of z^*. Conversely, given such a C, a null-space vector z^* whose support is C can clearly be constructed.

A column subset C such that $A_{\bullet C}$ has $|C| - 1$ non-zero rows R and such that $M(A_{RC_j}) = |R|$ for all $j \in C$ is called a **circuit** of A. (It is called this because it is a circuit of the matroid generated by the columns of A. A circuit of a matroid is a minimal dependent set; see Welsh (1976).) Thus if NSP has a polynomial algorithm, there is also a polynomial algorithm for the following problem:

The Minimum Circuit Problem (MCP): Given an $m \times n$ sparse matrix A, find a minimum cardinality circuit of A.

The size of a minimum circuit of A is called the **girth** of A.

Theorem 3.3.6: MCP is NP-Complete, and thus NSP is NP-Hard.

Proof: This proof is due to Stockmeyer (1982). The problem that we shall reduce to MCP is

The m-Clique Problem: Given an undirected graph $\mathcal{G}$, determine whether $\mathcal{G}$ has a **clique** of size m (a clique is a node-induced complete graph; see Bondy and Murty (1976), Section 7.2).

A proof that the m-Clique Problem is NP-Complete is referenced in Garey and Johnson (1979), Problem [GT19].

Given a graph $\mathcal{G}$ with v vertices and e edges, construct a $\left(v + \binom{m}{2} - m - 1\right) \times e$ sparsity pattern $A(\mathcal{G})$ as follows. Index the first v rows of $A(\mathcal{G})$ with the vertices of $\mathcal{G}$, and its columns with the edges of $\mathcal{G}$. Make $a_{i,\{j,k\}} \neq 0$ when $i = j$ or $i = k$, so that the first v rows of $A(\mathcal{G})$ have the same sparsity pattern as the vertex-edge incidence matrix of $\mathcal{G}$. For row $i > v$, set $a_{ie} \neq 0$ for all e.

Suppose that $A(\mathcal{G})$ has a circuit C with $|C| = c < \binom{m}{2}$. Let $d = \binom{m}{2} - c$, so that $d > 0$. Since C is a circuit, $A_{\bullet C}$ has $c - 1$ non-zero rows. Because the $\binom{m}{2} - m - 1$ non-vertex rows are among these, $A_{\bullet C}$ has $(c-1) - \left(\binom{m}{2} - m - 1\right) = m - d$ non-zero vertex rows. Denote the set of such rows by R, and denote by f the number of edges in the subgraph of $\mathcal{G}$ induced by the vertices in R. Since $|R| = m - d$ vertices can induce at most $\binom{m-d}{2}$ edges, $f \leq \binom{m-d}{2}$. But surely all the edges in C are among those induced by R, so that $f \geq c = \binom{m}{2} - d$. Putting these inequalities together yields $\binom{m}{2} - d \leq e \leq \binom{m-d}{2}$. However, it is easy to show that when $m > 3$ (which can be assumed without loss of generality), $\binom{m}{2} - d > \binom{m-d}{2}$.

Thus every circuit C of $A(\mathcal{G})$ must satisfy $|C| \geq \binom{m}{2}$. Suppose that $\mathcal{G}$ has an m-clique, say on the vertices in the set R (so that $|R| = m$), and with the $\binom{m}{2}$ edges in the set C. Since $|R|$+(number of non-vertex rows) $= \binom{m}{2} - 1 = |C| - 1$, it is easy to verify that C is a circuit.

Conversely, suppose that $A(\mathcal{G})$ has a circuit of size $\binom{m}{2}$, and let R be the m non-zero vertex rows in $A_{\bullet C}$. Let f again be the number of edges in the subgraph of $\mathcal{G}$ induced by R. As above, it must be true that $f \geq |C| = \binom{m}{2}$. But m vertices can induce at most $\binom{m}{2}$ edges, implying that $f \leq \binom{m}{2}$. Hence $f = \binom{m}{2}$, and the vertices in R are an m-clique.

Thus $\mathcal{G}$ has an m-clique if and only if the girth of $A(\mathcal{G})$ is $\binom{m}{2}$. If there were a polynomial algorithm for MCP it could be used to determine the girth of $A(\mathcal{G})$, and thereby determine whether $\mathcal{G}$ has an m-clique. But the m-Clique Problem is NP-Complete, and so MCP must also be NP-Complete. Therefore solving NSP is NP-Hard, since (as shown above), solving NSP also solves MCP. □

This theorem establishes the somewhat surprising result that, under the assumption (MP), NSP is NP-Hard even though SP' = $(\text{NSP})^2$ and SP has a polynomial algorithm. Hence complexity is not preserved under taking "square roots."

This analysis also has a connection with a different area of complexity research. Hausman and Korte (1981) have investigated the relative power of various matroid oracles in order to improve understanding of the complexity of matroid algorithms. They have shown that a girth oracle is strictly stronger than any other matroid oracle studied. It was previously known that there is a polynomial girth oracle for graphic matroids (see Itai and Rodeh (1978)), but no polynomial girth oracles have been discovered for any more complicated classes of matroids. The matroids generated by the columns of sparse matrices satisfying (MP) are **transversal** matroids (see Welsh (1976), Section 7.3). Transversal matroids are one of the simplest classes of matroids besides the class of graphic matroids, yet Theorem 3.3.6 shows that it is extremely unlikely that there is a polynomial girth oracle for transversal matroids. Perhaps this is why girth oracles are so powerful.

3.4. Practical Algorithms for SP

We now consider using the algorithms of Section 3.3 to process real matrices. The behavior of the algorithms when A does not have full rank and when it does not satisfy (MP) is investigated. Various modifications to the algorithms that decrease their running time in practice are also discussed.

3.4.1. Processing Rank-deficient Matrices

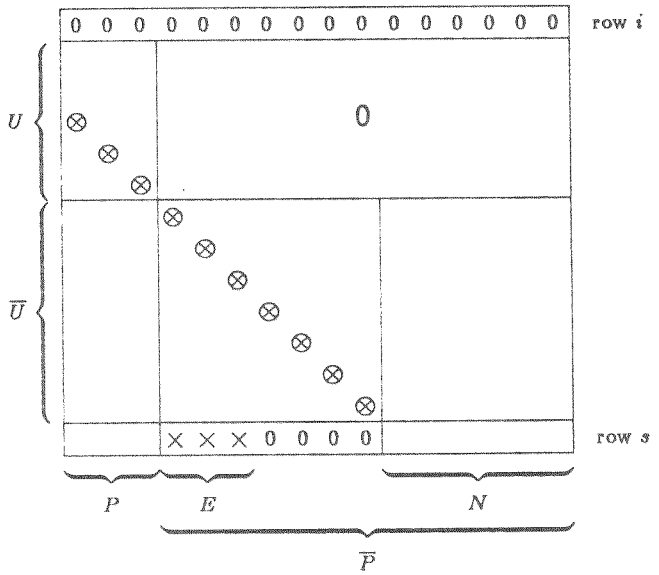
The first step in developing a more practical algorithm than PA or SA is to drop the assumption that rank $A = m$. The object of solving SP' is to find a sparser matrix $\hat{A}$ that spans the same row space. Therefore, when A is rank-deficient an algorithm should select a row basis for A, delete the remaining dependent rows, and use PA or SA to make the row basis optimally sparse. The next theorem shows that while (MP) still holds, the answer obtained is independent of the choice of basis.

Theorem 3.4.1: Both PA and SA produce the same final number of non-zeros when applied to any maximum independent subset of rows of A.

Proof: By Theorem 3.3.4, it suffices to prove the theorem for PA. Any two bases are connected by a sequence of row swaps, and hence it is sufficient to consider two bases that differ only by a swap.

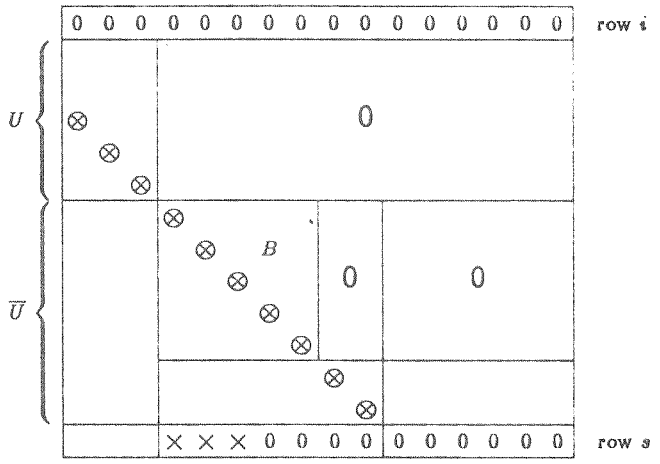
Let B_1 be a subset of rows which is a basis for A, and let r and s be rows with $r \in B_1$, $s \notin B_1$ such that $B_2 = B_1 \cup \{ s \} \setminus \{ t \}$ is also a basis for A. We shall show that PA produces the same final number of non-zeros when applied to $A_{B_1 \bullet}$ and $A_{B_2 \bullet}$. Since (MP) is still assumed, $\operatorname{rank} A = M(A) = |B_1| = |B_2|$.

Let $i \in B_1 \setminus \{ r \}$, and let l_k be the size of a minimum cover of the i^{th} zero-section in $A_{B_k \bullet}$, $k = 1, 2$. By (3.2.1) it suffices to show that $l_1 = l_2$. Define $U, \overline{U}$ and $P, \overline{P}$ so that they partition the rows and columns of the i^{th} zero-section of $A_{B_1 \bullet}$ as before, and hence $U = U_i^*$ (of $A_{B_1 \bullet}$), etc. Recall that $L_1 = \overline{U} \cup P$ is the standard minimum cover of the i^{th} zero-section of $A_{B_1 \bullet}$ by lines, so that $l_1 = |L_1|$. The i^{th} zero-section of $A_{B_1 \bullet}$ with a maximum matching M and row s adjoined must look like



By Fact 1 of Theorem 3.3.5, M can be extended to a perfect matching $\overline{M}$ of all of $A_{B_1 \bullet}$ that uses the same columns as M. Hence it can be assumed without loss of generality that M is part of a row-perfect matching of $A_{B_1 \bullet}$. Thus row s must be zero in columns in N (the **unmatched** columns), for otherwise $\overline{M}$ could be trivially extended to row s, contradicting the fact that B_1 is a basis.

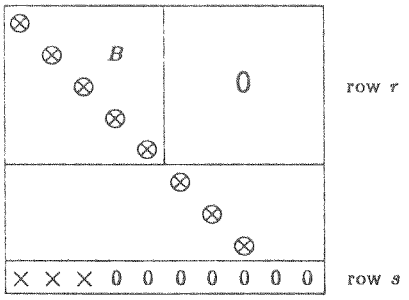
Define E to be the columns in $\overline{P}$ where row s is non-zero, as shown above. Let $\hat{M}$ be M restricted to $A_{\overline{U}\overline{P}}$. Try to extend $\hat{M}$ by labelling in the submatrix $A_{\overline{U} \cup \{ s \}, \overline{P}}$. Since $A_{U \cup \{ i \}, \overline{P}}$ is zero, $\hat{M}$ cannot be so extended, for otherwise $\overline{M}$ could be extended. Define the submatrix of labelled rows and columns of $A_{\overline{U}\overline{P}}$ to be B; clearly $E \subseteq \{ \text{columns of } B \}$, and by the properties of the labelling process, B must be square. Note also that the rows of B must be zero in the columns of $\overline{P} \setminus \{ \text{columns of } B \}$, for otherwise more rows and columns would have been labelled. Thus the picture has now become



Define $L_2 = L_1 \cup \{\text{columns of } B\} \setminus \{\text{rows of } B\}$. By the above remarks, L_2 must cover $A_{B_1 \cup \{s\},\bullet}$, and so it also covers $A_{B_2\bullet}$. Note that $|L_1| = |L_2| = l_1$. Since L_2 is a cover of the i^{th} zero-section of $A_{B_2\bullet}$, l_2 can be at most l_1. Now repeat the argument, reversing the roles of B_1 and B_2, and of r and s, obtaining $l_1 \leq l_2$. Thus $l_1 = l_2$, and consequently PA produces the same result on row i with either B_1 or B_2.

It remains only to show that PA produces the same final number of non-zeros for r in $A_{B_1\bullet}$ as it does for s in $A_{B_2\bullet}$. Note that row r might have a different *starting* number of non-zeros than row s. It is only claimed that the final answers have the same number of non-zeros; if, say, row r has more initial non-zeros than row s, PA must eliminate more non-zeros from $A_{B_1\bullet}$ than from $A_{B_2\bullet}$ to obtain the same final number of non-zeros.

Let M be a perfect matching in $A_{B_1\bullet}$, and adjoin s to $A_{B_1\bullet}$. Now try to extend M in $A_{B_1\cup\{s\},\bullet}$ by labelling as above, starting at s, again obtaining a square submatrix B of $A_{B_1\bullet}$ such that the rows of B are zero outside the columns of B. Thus the picture must look like



Since s can be exchanged for r while maintaining a matching of size $|B_1|$, r must be among the rows of B. But B has a perfect matching, so that PA can clearly use all the rows $k \neq r$ in B to eliminate all but one non-zero from row r, and this use of rows is optimal. By reversing roles again, the transformed row s in $A_{B_2\bullet}$ must also contain only one non-zero. Thus PA produces the same result using either B_1 or B_2. □

This proof shows, however, that $|U_i^*|$ can be different for B_1 and B_2, and so one basis might be best in terms of requiring the least amount of work. Unfortunately, we know of no way to determine such a basis.

3.4.2. Processing Matrices without (MP)

The next step towards practicality is the major one of dropping (MP). Though the use of (MP) as a tool to derive an algorithm is quite justified, it does not follow that (MP) is actually satisfied by most real matrices. Indeed, the results in Section 3.5 show that none of the real matrices tested satisfies (MP). Now (MP) is formally renounced as an assumption. To simplify matters at first, assume once again that $\operatorname{rank} A = m$.

Without (MP), as was discussed in Section 3.3, PA is unuseable in its present form. To reiterate, it is unuseable because PA assumes that any square submatrix selected by a fixed row-perfect matching is

non-singular, which is no longer necessarily true. In contrast, SA has great freedom in choosing the G_i. The advantage of SA over PA is that SA can choose G_i by a *numerical* criterion rather than a combinatorial one.

To see how SA chooses G_i in practice, first note that since A (or a basis of A) starts out with full row rank, and since A is multiplied by a sequence of non-singular transformations as SA progresses, at any point in the execution of SA, and for any row subset U, $\operatorname{rank} A_{U*} = |U|$. Thus A_{U*} must always have *some* column subset G such that A_{UG} is non-singular. At the appropriate point of SA, A_{U*} is passed to a subroutine that can numerically select a G so that A_{UG} is non-singular. A subroutine that can perform sparse LU factorizations of rectangular matrices is ideal. In the experimental implementation described in Section 3.5, a state-of-the-art Harwell black-box subroutine called MA28 is used for this purpose (see Duff (1977)).

A further benefit of choosing G numerically by an LU-factorization is that it neatly combines Steps 2 and 3 of ORA_i. That is, since the factorization routine selects the G such that the LU-factorization of A_{UG} was was computed, solving for λ at Step 3 of ORA_i becomes quite easy. Also, a good factorization routine like MA28 chooses G so that A_{UG} is fairly well-conditioned. This property gives some assurance that the reduced A is not much worse conditioned that the original A.

The above method of implementing SA enables it to reliably process real matrices. When SA is applied to a real problem A, we would ideally like to guarantee that it reduces the number of non-zeros of A at least as much as if A *did* satisfy (MP). That is, for a given sparsity pattern A, there is a well-defined reduction in non-zeros r possible by either PA or SA, independent of the values of non-zero entries of A, as long as they satisfy (MP) (indeed, it is possible to run either PA or SA on a sparsity pattern totally combinatorially, without doing any numerical operations whatsoever). Not satisfying (MP) means that unexpected cancellation can happen, and it seems that such cancellation could only help. Hence it should be possible to show that at least r non-zeros are eliminated.

However, proving such a guarantee is somewhat subtle. Consider the full-rank matrix

$$A = \begin{pmatrix} 1 & 3 & 0 & 5 & 5 \\ 2 & 1 & 4 & 0 & 0 \\ 0 & 3 & 0 & 5 & 5 \end{pmatrix}.$$

The Sequential Algorithm chooses $U_1 = \{3\}$, and could choose $G_1 = \{2\}$. The associated transformation unexpectedly zeros out columns 4 and 5 of row 1. If row 2 is processed using the new row 1, SA chooses $U_2 = \{1\}$. But the parallel $U_2^* = \emptyset$, which does not contain U_2 as required by the induction hypothesis of Theorem 3.3.4. Since the performance of SA depends on the hypothesis, it can not be guaranteed that SA eliminates as many non-zeros as the ideal. (A close reading of the proof of Theorem 3.3.4 reveals that this difficulty can arise only when $\operatorname{rank} A_{RZ} < M(A_{RZ})$ for some zero-section; in the second zero-section of this example, $\operatorname{rank} A_{RZ} = 1 < 2 = M(A_{RZ})$.)

A simple trick avoids this difficulty. As SA executes, at each step it can recognize where non-zeros are expected to occur for subsequent steps. Unexpected cancellation can be recognized in two ways. First, if the current row is i and $a_{ij} \neq 0$, $j \notin G_i$, but $\hat{a}_{ij} = 0$, SA has a **lucky hit**. Second, if $a_{ij} = 0$ and $j \in P(U_i) \setminus G_i$ (the expected fill-in columns), but $\hat{a}_{ij} = 0$, then SA has a **lucky non-fill-in**. When unexpected cancellation occurs, SA can put a **phantom** non-zero in that entry of the matrix (a zero that is treated as if it were a non-zero). That is, subsequent matchings are performed as if no unexpected cancellation ever took place, although SA keeps track of which "non-zeros" are really zeros. As long as A initially has full row rank, the numerical operations can never create a dependence among the rows. Thus SA can always find a G so that A_{UG} is non-singular, even with phantom non-zeros. When SA is modified by using phantom non-zeros, it is called the **Safeguarded Sequential Algorithm (SSA)**.

Theorem 3.4.2: SSA eliminates at least as many non-zeros from any full-rank matrix A as it would if A satisfied (MP).

Proof: The proof of Theorem 3.3.4 becomes valid once again with SSA, which yields the guarantee. □

Of course, it is possible to apply SA on real problems without safeguarding by keeping phantom non-zeros, but the guarantee of Theorem 3.4.2 is lost. Letting SA "know" about more zeros by removing the safeguard might give it freedom to produce greater reductions, but the experiments in Section 3.5 show no clear advantage for either SA or SSA.

Now we drop the assumption that $\operatorname{rank} A = m$ once again. The following example shows that Theorem 3.4.1 is no longer true:

$$A = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ -1 & 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 0 & -1 \end{pmatrix}.$$

Though $M(A) = 4$, $\operatorname{rank} A = 3$, since the sum of the rows is zero. If the first row is dropped to get a full-rank submatrix, then neither SA nor SSA eliminates any non-zeros from A, for a final total of 7 non-zeros. If the last row is dropped, then once again neither SA nor SSA reduces any non-zeros from A, for a final total of 8 non-zeros. We know of no way for choosing a basis of A that maximizes the final number of non-zeros. We conjecture that the difference between bases is negligibly small in practice, so that this issue would be a major concern. The truth of this conjecture has not yet been empirically tested.

3.4.3. Miscellaneous Modifications to SA

The first step in any implementation of SA or SSA must be the determination of a row basis for A. As discussed above, a routine such as MA28 is ideal for this task. If a row basis is determined through an LU-factorization as in MA28, an important side benefit can be realized.

Note that an LU-factorization picks out a square submatrix A_{RC} of A such that $|R| = |C| = \operatorname{rank} A_{RC} = \operatorname{rank} A$. As SA or SSA processes $A_{R\bullet}$, the partially transformed A_{RC} remains non-singular just as $A_{R\bullet}$ does. Thus, when the algorithm needs to search $A_{U\bullet}$ for a non-singular submatrix A_{UG}, it can restrict its search to A_{UC}. Since $|C|$ is usually much smaller than n, such a restriction can lead to large savings in time if A_{UC} instead of $A_{U\bullet}$ is factored to find G. Of course, restricting the columns in which G_i can occur also restricts the freedom of the factorization routine to choose a well-conditioned G_i, but since A_{RC} is chosen to be reasonably well-conditioned, the restricted G_i should not be much worse conditioned than the unrestricted G_i. This modification is called the **restricted column option**. The results in Section 3.5 report on its performance.

There is another modification that can speed up the combinatorial parts of the algorithms. All of the combinatorial effort of the algorithms involves computing maximum cardinality matchings in zero-sections. Instead of starting each such matching from the empty matching, it is faster to initialize them as follows. Start by finding a one-time fixed maximum matching for A, call it $\mathcal{M}$. Then initialize the matching in the i^{th} zero-section with $\mathcal{M}$ restricted to the columns of the zero-section. An entry in $\mathcal{M}$ might be eliminated at some point during execution of the algorithm. If this should happen, a single augmentation of $\mathcal{M}$ before the next iteration restores $\mathcal{M}$ to a maximum matching. This method of initialization is called **warm-start matching**.

With warm-start matching a good bound on the combinatorial running time of SA and SSA can be derived. Let ν be the number of non-zeros of A, which can be assumed to be greater than n. Finding the original $\mathcal{M}$ takes $O(\sqrt{m+n}\nu)$ time (see Papadimitriou and Stieglitz (1982), Theorem 10.3). Copying the columns of $\mathcal{M}$ into a starting solution for the i^{th} zero-section takes $O(m)$ time, and hence copying takes $O(m^2)$ overall. After copying $\mathcal{M}$ into the i^{th} zero-section, each remaining unmatched row in the zero-section matches to a column *outside* the zero-section under $\mathcal{M}$. The number of initially unmatched rows in the i^{th} zero-section is at most the number of non-zeros in row i of A, and so the total number of unmatched rows is at most ν. Each unmatched row can lead to at most one augmentation of a matching, so that the total number of zero-section augmentations is at most ν. An augmentation is an $O(\nu)$ process, making the total time spent in zero-section matching $O(\nu^2)$. Finally, each entry of $\mathcal{M}$ can be eliminated at most once (when its row is processed); therefore, a single augmentation of $\mathcal{M}$ might be needed at most m times. Again each augmentation takes at most $O(\nu)$, for total time $O(m\nu)$ spent repairing $\mathcal{M}$. Therefore the total time bound for the combinatorial running time of SA and SSA with warm-start matching is $O(\nu^2)$.

It is more difficult to obtain an accurate bound on the numerical running time of SA and SSA. In the worst case, even with the restricted column option, the algorithms have to factor and solve a linear system of size $O(m)$ for each of m rows (plus once at the beginning to obtain a basis for A). Since factoring and solving one such system is bounded by $O(m^3)$, an overall numerical bound is $O(m^4)$. However, since A is sparse, the systems to be solved are also sparse. An efficiently implemented sparse equation solver like MA28 tends to

solves such systems in time $O(\nu)$ (see Duff (1977), Table 3), which would give the much better overall bound of $O(m\nu)$ for numerical operations.

As a final observation on applying the algorithms to practical problems, note that linear constraints are usually not presented as a system of equations as in (3.1.3), but as a mixture of inequalities and equations. Inequalities can be converted to equalities by adding a slack variable, so that the constraints are then in a form which is suitable for the algorithms. However, there is another modification that can be used to save time when inequalities are present.

First, note that an inequality row i can never be used by another row, because its slack column (a unit vector), is in the zero-section of every other row. It can be assumed without loss of generality that the maximum matching in each such zero-section includes the slack entry. Since all other entries in the slack column are zero, there is no way for row i to be labelled during the matching in that zero-section. Thus row i is in the U of no other row. (But row i can itself be reduced by other rows during the algorithm.) Indeed, whenever a row i has a non-zero in a column which is a unit vector, row i is never used by any other row. Thus, if A contains an embedded identity matrix, the algorithm does not reduce A at all.

Given that the slack columns do not participate in the matchings, it is more efficient to create a single "phantom" column (column 0), instead of many slack columns. In using warm-start matching, all the inequality rows can be permanently matched to column 0 before finding $\mathcal{M}$. If column 0 is artificially included among the columns of every zero-section, all the inequality rows initially match to column 0 in the zero-section since $\mathcal{M}$ is used for an initial matching. Since column 0 contains no non-zeros, it is never labelled during computation of a matching in a zero-section. Therefore, none of the inequality rows is ever labelled either, so that the inequality rows are automatically not used. Furthermore, since the inequality rows stay matched throughout the process, this strategy effectively reduces the size of the matching problem in each zero-section.

More importantly, the inequality rows can be excluded from A in determining its rank at the beginning, because inequality rows always have full rank and are never used anyway. Therefore, the square full row rank submatrix A_{RC} obtained from the initial LU-factorization excluding the inequality rows is suitable for using the restricted column option, and C is even smaller. These strategies for treating inequalities are called **phantom slacks**. By using phantom slacks numerical execution time can be even further reduced.

The disadvantage of inequalities is that, by decreasing the number of rows available for use, they lead to smaller reductions of non-zeros. The extreme case is that when all constraints are inequalities, the algorithm is not able to eliminate any non-zeros. A general rule of thumb for using SA and SSA is that the higher the proportion of equality constraints, the better.

Finally, potential implementors are reminded that as the rows of A are transformed, the same transformations must be applied to the right-hand side(s) b. However, any `RANGES` or `BOUNDS` (see Murtagh (1981), Section 9.2) do not need to be changed.

3.5. Computational Results

In this section we shall first describe the implementation of an experimental version of the algorithms of Section 3.4. Some preliminary computational results from the implementation are then discussed.

3.5.1. An Experimental Implementation of the Algorithm

The experimental implementation of SA and SSA is a `FORTRAN` program called SPARSER. The program reads A in industry-standard `MPS` format (see, *e.g.*, Murtagh (1981), Section 9.2), processes it by one of several variant algorithms (depending on its input parameters), and outputs the reduced A in `MPS` format if desired. The `MPS` input routine is a modified version of the routine used by MINOS (see Saunders (1977)), which uses Brent's (1973) version of double hashing to reduce time spent in row look-up.

The two biggest tasks for SPARSER are computing maximum cardinality matchings in various submatrices of A, and computing the LU-factors of various rectangular submatrices of A (and solving the resulting square subsystems). The matching is performed by a modified version of a depth-first search look-ahead technique, as described in Gustavson (1976). Though this algorithm has poorer worst-case performance than the Hopcroft and Karp (1973) algorithm, creators of sparse matrix software have empirically observed that its average performance on typical real problems is better than other algorithms. The lack of

a realistic random model of sparse matrices has prevented any attempt to provide theoretical support for this observation.

As mentioned in Section 3.4, the numerical tasks in SPARSER are handled by MA28, a package of sparse matrix LU-factorization and linear equation-solving routines written by Duff at Harwell (see Duff (1977)). The advantage of using MA28 for SPARSER is that it can process rectangular matrices. This feature is important because the only practical way to choose column subset G from A_{U*} (or from A_{UC} if using the restricted column option of Section 3.4) is to factor all of A_{U*}, which lets G be dynamically specified by the choice of pivot columns. MA28 uses a hybrid method for choosing its pivots based on a pivot stability parameter U which controls the trade-off between stability and sparsity. Setting U = 1.0 makes MA28 chose pivots purely on the basis of stability; choosing U = 0.0 makes it choose pivots purely on the Markowitz sparsity criterion (see Markowitz (1957)). The value U = 0.1 is recommended in Duff (1977) and was used in all tests reported herein. The value of U can be set by an input to SPARSER.

MA28 has other parameters of interest to designers of SP algorithms. When MA28 parameter LBLOCK is .TRUE., MA28 block-triangularizes its input (finds the DM decomposition of a square submatrix) before it is factored. When the chosen submatrix decomposes into relatively small blocks, time can be saved since it is cheaper to factor many small matrices rather than one big one. However, decomposing a submatrix can be dangerous when the matrix is rectangular.

For example, within SPARSER, the matrix A_{U*} (or A_{UC}) (which is known to have rank $|U|$) is input to MA28. When MA28 performs a block-triangularization, it first finds a maximum matching M in the matrix, which must also be of size $|U|$. The difficulty is that MA28 forces itself to factor the size $|U|$ *square* submatrix B induced by M, and it can happen that $\operatorname{rank} B < |U|$. This possibility once again illustrates the reason why SA is preferred over PA: real matrices do not always have full numerical rank when they have a perfect maximum matching. A retry function within SPARSER overcomes this difficulty by deleting the column that MA28 indicates is dependent and re-factoring, but doing this slows SPARSER down. Thus, although SPARSER includes a way to change LBLOCK, it is strongly recommended that block-triangularization be disabled.

Another MA28 parameter of interest controls the solution routine and is called MTYPE. If the input matrix is B and the right-hand side is b, then MTYPE controls whether $Bx = b$ or $xB = b$ is the system to be solved. Since SPARSER needs the solution of $\lambda_U A_{UG} = -A_{IG}$, it would seem that MTYPE should be defined so that the second option is always taken. However, SPARSER subsumes MTYPE into a parameter of its own that controls which one of A_{U*} or A_{U*}^T is input to the factor routine, and that selects the appropriate value of MTYPE accordingly. This option allows experimentation to determine whether it is faster to factor rectangular matrices with the smaller or the larger dimension first within SPARSER.

Two parameters particular to SPARSER are relevant here. The first specifies which algorithm is used to process A. When describing algorithms, "combinatorial" means that an algorithm is applied formally to the sparsity pattern of A, without performing any numerical operations. A combinatorial algorithm is of use only as a fast way to determine the performance of an algorithm on a given matrix; since only the sparsity pattern of the reduced matrix is correct, not the numerical values, the reduced matrix cannot serve any further useful purpose. By contrast, "numerical" means that numerical operations are performed, so that the reduced A is equivalent to the input A.

With this understanding of terms, SPARSER allows four algorithmic options: combinatorial PA, combinatorial SA, numerical SSA, and numerical SA. The first two options are mainly used verify the correctness of SPARSER as it evolves; by Theorem 3.3.4 they should always give the same final number of non-zeros. They are also useful for quickly checking a new matrix, as noted above. The third alternative is SSA as described in Section 3.4, with the guarantee of Theorem 3.4.2. The fourth alternative is SA without the phantom non-zeros safeguard of SSA that allows Theorem 3.4.2 to be applied.

All versions of the algorithm use warm-start matching and phantom slacks as described in Section 3.4, but the restricted column option is a user-selectable option. The trade-off between decreased stability and smaller execution time with the restricted column option can then be tested.

Four parameters govern how A is processed by SPARSER, each with two values: SA can be run with safeguarding or not, block-triangularization or not, factoring A or A^T, and using the restricted column option or not. Thus there are 16 variant algorithms available through SPARSER.

Red. Col. Opt.?	Block Δ-ize?	Factor Transp. A?	Use Safe-guard?	Total Time	Total Gain	Max Grow	Total Used	Max Used	No. Remat-chings
N	N	N	N	2.32	644	194	751	35	7
			Y	2.30	645	194	753	35	6
		Y	N	2.41	645	194	761	37	7
			Y	2.41	646	194	763	37	6
	Y	N	N	3.32	648	1.55	751	35	11
			Y	3.31	649	1.55	753	35	10
		Y	N	3.26	641	194	750	35	10
			Y	3.26	642	194	752	35	9
Y	N	N	N	1.75	642	194	751	35	9
			Y	1.75	643	194	753	35	8
		Y	N	1.83	642	194	756	36	9
			Y	1.80	643	194	758	36	8
	Y	N	N	2.34	642	6.72	755	36	12
			Y	2.39	643	6.72	757	36	11
		Y	N	2.35	641	194	750	35	10
			Y	2.39	642	194	752	35	9

Table 3.5.1

3.5.2. Testing the Variations of SPARSER

Each of the 16 variations of SPARSER was used to process a linear programming matrix called BANDM. These runs held all other minor parameters fixed, and in particular set all print flags to their lowest level and disabled MPS output of the reduced matrix, to allow the execution time to reflect only the basic processing. The matrix of BANDM has $m = 305$, $n = 472$ and contains 2494 non-zeros, a density of 1.73%. It has 305 equality rows, 100% of the total. The combinatorial number of non-zeros that can be eliminated from BANDM (the guarantee of Theorem 3.4.2) is 633, which is 22.8% of the total non-zeros (an impressive figure).

The results of these runs are displayed in Table 3.5.1. All testing was performed while running SPARSER interactively on VM on an IBM 3081 at the Stanford Linear Accelerator Center. The "total time" is in CPU seconds while running SPARSER interactively on VM. The "total gain" column adds the lucky hits and lucky non-fill-ins (each one a proof that (MP) is not satisfied) to the guaranteed combinatorial gain of 633 non-zeros. Column "max grow" gives the maximum value over all rows of an MA28 output parameter called GROW. The value of GROW estimates the extent to which the numerical operations on a row cause the entries of A to "blow up" numerically, and so provides an indirect measure of the stability of the reduced matrix relative to the original matrix (see Duff (1977), pp. 17–18). Column "total used" gives $\sum_i |U_i|$, an indication of the sizes of the linear systems that were solved within SPARSER, and "max used" gives $\max_i U_i$, which helps to indicate whether most of the gain is coming from a few rows ("max used" nearly as big as total used) or is spread out ("max used" very small). Finally, "no. rematchings" reports how many times out of the 305 equality rows an entry of the fixed matching was hit, necessitating repair.

Starting from the less important conclusions that can be drawn from Table 3.5.1, there seems to be little significant difference among the 16 variations of the algorithm on any of the last five columns, with two

possible exceptions. First, "max grow" varies from 1.55 to 194, which might indicate that some variations are inherently more stable than others. However, our experience has been that a value of 194 for "max grow" is no cause for concern, and we conjecture that the lower values of "max grow" resulted more from luck than any fundamental differences among variations. Second, there appears to be a slight trend for the number of rematchings to increase when using block triangularization. However, this possible deficiency pales beside the other flaws of block-triangularization, as shown by the results below.

There seems to be only random variation in total gain (from 641 to 649), the stated objective. A pattern might possibly emerge if the 16 variations were run on a different matrix for which lucky gains were a bigger part of total gain than for BANDM. It is shown later that BANDM has an atypically small amount of lucky gain. This issue will be tested further in the future. The only remaining criterion by which to judge the variations is their execution times. There *is* a definite spread of time; the longest time is 3.32 seconds, almost twice the shortest, 1.75 seconds.

Safeguarding or not makes very little difference in time according to Table 3.5.1, though again this might change with a matrix with more lucky gain.

Factoring submatrices of A in their normal (as opposed to transposed) form appears to be faster for this application, despite the opposite advice of the author of MA28. That is, factoring submatrices of BANDM with the smaller (row) dimension first was faster, though Duff (1977, p. 28) suggests putting the larger dimension first. The apparent exception to this observation is that without the reduced column option and with block-triangularization, in which case factoring the transpose is faster (see Table 3.5.1). This anomaly seems to be due to the fact that in both of the cases when submatrices were normally factored, block-triangularization caused difficulties by selecting singular matrices four times. The retry routine then caused four extra linear systems to be solved, tipping the time balance to factoring the transpose, which had no bad luck with block triangularization.

Besides its other flaws mentioned above, using block-triangularization on BANDM makes SPARSER run more slowly. This result is initially surprising, since block-triangularization is supposed to speed up solving equations. An explanation for this behavior can be found in the "total used" column of Table 3.5.1. Since BANDM has 305 rows, and "total used" is roughly 755 rows, an average of less than 2.5 other rows are used per row of BANDM. Thus the linear systems passed to MA28 are already quite small, so that the block triangularization code adds an overhead greater than any possible savings. Even the supposed virtue of block triangularization is a flaw, therefore its use in SPARSER is discouraged more severely than before.

Finally, using the restricted column option leads to a big decrease in time (which is scarcely any surprise). Without it, MA28 is passed a matrix whose largest dimension is $n = 472$. Using the restricted column option, the largest dimension drops to the rank of the equality rows, which is 305 for BANDM, which is a decrease of 35.4%. Since we suspect that numerical operations dominate SPARSER's processing time, such a large decrease is bound to have a big effect on running time.

With these results in mind, the rest of the tests were conducted with no block triangularization, factoring A in normal form, using the restricted column option, and using safeguarding. The last choice was made only because it is better to be safe than sorry.

3.5.3. Testing SPARSER on Real Matrices

The next objective in testing SPARSER was to apply it to a variety of real problems to see how well it does in practice. We obtained 23 linear programming problems from Harlan Crowder of the IBM T. J. Watson Research Laboratory in order to investigate the performance of SPARSER. They were selected solely on the basis of having a high proportion of equality constraints, and range in size from AFIRO (which is 27×32 with 83 non-zeros) to AIRFSTAR (which is 311×3637 with 10,513 non-zeros).

The results of running SPARSER on these 23 problems are displayed in Table 3.5.2. The "rows" and "columns" figures quoted include only those **relevant** to SPARSER (*i.e.*, objective rows and right-hand sides are excluded), and "non-zeros" counts only those non-zeros in relevant rows and columns. The "% eq. rows" reports what percentage of the relevant rows are equalities; recall from Section 3.4 that this figure is potentially important in determining the performance of SPARSER. Three of the matrices have rank-deficiencies that effectively reduce the percent of equality rows in Table 3.5.2 (BOAC from 41.2 to 36.8; CRACPB1 from 62.2 to 62.0; BRANDY from 75.5 to 72.0); the higher figure is used in Table 3.5.1 since it can be

Name	# of Rows	# of Cols.	# of Non-Zs.	% Eq. Rows	Orig. % Den.	% Red. Non-Z.	Lucky Comb.	Total Time	Max. Grow
AFIRO	27	32	83	29.6	9.61	0.00	—	.08	—
BOAC	313	298	1659	41.2	1.78	0.00	—	1.97	—
PB5SOS2	271	1426	2648	82.7	0.69	0.00	—	1.15	—
WEYERHSR	41	107	706	97.6	16.09	.85	0.20	.23	2.82
STORESOS	68	590	2784	86.8	6.94	2.44	.58	.84	829
ADLITTLE	56	97	383	26.8	7.05	3.39	.08	.23	45
FRESNO	208	316	1791	47.1	2.72	3.46	.03	.55	10.6
L84MAV	113	1995	9126	99.1	4.05	4.95	3.07	5.62	0.0
L152D	93	1550	9862	98.9	6.84	7.16	6.76	2.28	0.0
L21LAV	108	1939	8779	99.1	4.19	7.34	5.08	2.32	1.0
CAPRI	271	353	1767	52.4	1.85	8.09	.01	.89	6.13
CRACPBI	143	572	4158	62.2	5.08	8.37	2.87	1.77	0.0
SHARE2B	96	79	694	13.5	9.15	8.50	.23	.25	65
L27LAV	146	2655	11,203	99.3	2.89	8.84	5.78	3.99	1.0
E226	223	282	2578	14.8	4.10	9.00	1.97	.58	1.0
SHARE1B	118	225	1151	75.4	4.37	13.47	.12	.55	1810
BRANDY	220	249	2148	75.5	3.92	14.15	.30	.95	45.5
AIRFSTAR	311	3637	10,513	99.0	.93	14.61	21.26	3.35	1.0
L152LAV	97	1989	9922	99.0	5.14	16.11	15.82	2.46	1.0
LP4L	85	1086	4677	98.8	5.07	17.96	9.12	1.50	1.0
L94MAV	93	1750	7294	98.9	4.48	21.02	15.85	5.49	0.0
BANDM	305	472	2494	100.0	1.73	25.78	.02	2.04	194
BEACONFD	173	262	3375	80.9	7.45	64.06	.80	1.05	136

Table 3.5.2

determined without running SPARSER. The "orig. density" is the density of the original matrix, based on relevant non-zeros, relevant rows and relevant columns.

The linear programs in Table 3.5.2 are listed in increasing order of "% decr. in non-zeros," which is the total **reduction** in non-zeros achieved by SPARSER (both combinatorial and lucky) as a percentage of relevant non-zeros. It ranges from zero for the first three matrices, to an astounding 64.06% reduction for BEACONFD. The "$\frac{\text{lucky}}{\text{comb.}}$" column lists the relative contribution of combinatorial gains and lucky gains to the total gain. For example, BANDM had a combinatorial gain of 633 non-zeros and a lucky gain of 10 non-zeros, for a ratio of $10/633 = .02$, the second-smallest value listed. The "total time" and "max grow" columns have the same meaning as in Table 3.5.1. The discrepancy in time for BANDM between Tables 3.5.1 and 3.5.2 is due to the fact that the two sets of runs which are summarized in the tables were done several months apart. Thus times are comparable within tables but not between tables.

The runs summarized in Table 3.5.2 produced other interesting statistics. First, the total number of rematchings over all 23 linear programs was only 44, with BEACONFD alone accounting for 23. Thus, having to repair the fixed matching in warm-start matching does not result in large additional overhead. On a related note, when the bound on the combinatorial running time of SPARSER with warm-start matching was computed in Section 3.4, a key fact that was used is that the number of unmatched rows (after copying the fixed matching) is bounded by the number of non-zeros. In practice, the number of unmatched rows averaged out as less than 12% of the non-zeros.

The total used for each linear program reveals that the average $|U_i|$ for the 23 problems was only 1.06. Comparing the "combinatorial gain" and "total used" figures shows that each used row leads to a gain of .92 non-zeros on average. Recall that, by Theorem 3.2.2, each used row can make a combinatorial gain of at most one non-zero. Although saved and fill-in columns must be taken into account to derive the algorithm, in these runs they are rare events.

Many of the "lucky/combinatorial" figures may seem surprisingly high, but examining "max used" versus "total used" reveals the explanation. For seven of the eight linear programs with "lucky/combinatorial" > 3 (all except for AIRFSTAR, where "max used" is 4), "max used" is equal or nearly equal to "total used". This indicates that almost all of the total gain is achieved at one row. All seven of these lucky linear programs (those whose name starts with L) have one nearly dense row. The dense row is the only one with a non-empty U_i, and the matrices are structured so that cancelling out some of the non-zeros in the dense row also luckily cancels out nearly all the rest.

A high value of "lucky/combinatorial" may be caused by other special structures as well. When using SPARSER in practice, a high value of "lucky/combinatorial" could indicate that a more specialized method might be more appropriate than SPARSER. Despite having no provision to exploit such structure, SPARSER achieved creditable reductions on the lucky matrices anyway.

The results in Table 3.5.2 show that SPARSER can significantly reduce many matrices. The degree of reduction does not appear to be predictable from percentages of equalities or of density. The running time of SPARSER seems to be quite modest. Its large value on the lucky matrices is dominated by the time needed to determine the rank of the equality rows. Since the lucky matrices have an unusually large number of columns relative to rows, MA28 is required to factor an apparently huge matrix. Choosing to factor the transpose of such matrices might prove to be faster. The "max growth" in the reduced matrices was quite small for most of the linear programs.

3.5.4. Optimizing Reduced Matrices

Encouraged by the success of SPARSER in reducing this set of matrices, the 15 problems with reductions of at least 5% (the ones below the heavy line in Table 3.5.2) were tested in comparative optimization runs. The optimization program MINOS (see Saunders (1977)) was chosen for these test runs. MINOS is a high-quality transportable FORTRAN routine for solving problems of the type (3.1.1). It uses state-of-the-art sparse matrix techniques and is in daily use on the SLAC computer for solving a large energy model linear program (see Dantzig et al. (1981)). Rather than having SPARSER pass an internal representation of the reduced matrix to MINOS, SPARSER output an MPS file of the reduced matrix which was then used as the input for MINOS.

It is not easy to compare the time used by MINOS to optimize an original versus a reduced problem. The Simplex Algorithm follows the same pivot sequence on both A and TA if T is non-singular, except possibly in Phase I. The reason is that when Phase I adds artificial variables to A and TA, it obtains $(A \quad I)$ and $(TA \quad I)$, which are no longer equivalent. Thus Phase I can follow a different pivot sequence on the reduced problem than on the original problem, which would result in different initial feasible bases for Phase II. The overall result is a different pivot sequence and a different number of iterations before optimality, making comparison difficult.

We have attempted to circumvent this problem by starting both the original and reduced problems with the same feasible basis. This basis is obtained by running MINOS on the original problem until the first feasible basis is obtained. Then, in theory at least, both the original and reduced problems follow the same pivot path to optimality, so that any time differences can be attributed solely to increased sparsity. A drawback to this approach is that there are fewer iterations over which the cost of running SPARSER can be amortized.

Also, it is important to know whether reduced problems have any bias towards taking more or fewer iterations than original problems. Before we became aware of the Phase I difficulty discussed above, several pairs of original and reduced problems were optimized starting from a (non-equivalent) crash basis. No consistent bias in iterations was observed, but more formally organized experiments are needed to determine whether this is holds in general.

The results of the comparative MINOS runs are summarized in Table 3.5.3. The "% redn." column is copied from Table 3.5.2. The "orig. time" and "reduc. time" columns give the total time, in seconds, for

Name	% Redn. in Non-Zeros	Orig. Time (sec.)	Reduc. Time (sec.)	% Redn. in Time (corr.)	% Redn. in Time (uncorr.)	Itn. Ratio (or/red)
L152D	7.16	8.42	7.8	3.98	5.06	1.04
L21LAV	7.34	9.81	9.38	4.18	4.94	1.00
CAPRI	8.09	2.04	2.00	1.25	2.19	1.00
CRACPB1	8.37	2.70	1.93	−7.39	−4.69	1.62
SHARE2B	8.50	.55	.53	8.10	9.57	.95
L27LAV	8.84	12.72	12.60	.10	1.07	1.00
E226	9.00	7.04	6.80	3.20	3.57	1.00
SHARE1B	13.47	1.71	1.61	5.21	6.36	1.00
BRANDY	14.15	2.13	1.98	6.16	7.92	1.00
AIRFSTAR	14.61	5.83	5.90	6.87	10.50	.88
L152LAV	16.11	7.95	6.84	.19	3.20	1.16
LP4L	17.96	3.80	3.32	5.94	8.84	1.07
L94MAV	21.02	6.36	6.28	9.17	11.65	.90
BANDM	25.78	4.31	3.68	13.87	15.76	1.00
BEACONFD	64.06	.90	.63	9.42	49.17	1.00

Table 3.5.3

MINOS to solve the original and reduced problems respectively, starting from the same initial feasible basis, with MINOS running as a batch job on VM.

The last column of Table 3.5.3 gives the number of iterations for the original problem divided by the number of iterations for the reduced problem. It shows that starting the original and reduced problems at the same feasible basis does not always produce the same number of iterations in practice; where they differ, there is no discernible bias favoring either the original or reduced problem.

Columns 5 and 6 of Table 3.5.3 give estimated percent reductions in the time required to bring the starting feasible basis to optimality. An estimate (derived from SPARSER) of time spent inputting the MPS file was subtracted from the times reported in columns 3 and 4, in order to make the comparison more nearly reflect actual differences in time per iteration (the actual time spent iterating in MINOS is not currently available). The time spent in MPS input depends on the size of the linear program. Since a matrix reduced by SPARSER can have considerably fewer non-zeros than the original matrix, inputting a reduced MPS file can take less time than inputting the original MPS file. The MPS input time used to calculate the percent reduction in column 5 has been "corrected" by the ratio of number of reduced non-zeros to the number of original non-zeros to try to account for this difference. The percent reduction on column 6 has not been so corrected.

More formally, denote the total original optimization time from column 3 of Table 3.5.3 by OT, the total reduced time from column 4 by RT, the MPS input time from SPARSER by IT, and the iteration ratio from column 7 by r. Then the value in columns 5 and 6 is

$$100 \cdot \frac{(OT - IT) - r(RT - f \cdot IT)}{OT - IT}$$

where f is the ratio of original to reduced non-zeros for column 5, and is 1 for column 6. We believe that the true percent reduction in time lies between the values in columns 5 and 6.

The results in Table 3.5.3 show that reducing the number of non-zeros in these problems does indeed reduce the time needed to optimize them. Though the results have a large amount of variability, as a rule of thumb it appears that the percent reduction in iteration time is about half the percent reduction in non-zeros. In this light, these results are encouraging since they bear out the hope that problems with sparser matrices can be optimized faster. Moreover, SPARSER is sufficiently effective at increasing sparsity that the CPU time noticeably decreased.

But it is discouraging to compare the difference in original and reduced MINOS times in Table 3.5.3 with the SPARSER times in Table 3.5.2. In no case is the SPARSER time smaller than the overall time saved in MINOS. Pre-processing matrices to reduce optimization time is not helpful if the time saved is smaller then the time spent in pre-processing.

There are several factors that convince us that our algorithm will eventually prove to be practically useful in spite of its apparently unhelpful behavior in these experiments. One factor is that the linear programs that were tested are relatively small and solve relatively quickly. Another factor is that because of the way the experiments were set up, the reduction in time counts only the Phase II iterations, which typically are about half of the total iterations from a cold start.

The time taken by the algorithm should grow more slowly with increasing problem size than the time taken by an optimization routine. For example, Hillier and Lieberman (1974), p. 181, state that the solution time of linear programs is usually $O(m^3)$, whereas, as stated in Section 3.4, SPARSER uses only $O(m^2)$ time. Thus SPARSER would perform better on larger problems. Also, pre-processing with the algorithm is a fixed cost that saves time at every iteration of an optimization, and hence would be of higher utility on more difficult problems that take relatively more iterations. For example, the lucky linear programs are actually integer programs. Solving integer programs typically involves solving the associated linear program many times, in which case the algorithm would be more useful.

The current, experimental version of SPARSER spends some time debugging itself and accumulating statistics on its performance. A more streamlined implementation would be faster. Also, outputting a reduced MPS file from SPARSER for input to MINOS is inefficient. A more realistic implementation would integrate SPARSER into MINOS, thereby eliminating the unnecessary file-handling time.

On balance, the tests show that increasing sparsity is possible and that it reduces the time needed to solve a problem. It remains to be shown (by more extensive tests) whether our algorithm is a practical way of increasing sparsity. The test results so far are not overwhelmingly encouraging, but they do suggest that a streamlined version of SPARSER may prove useful.

3.6. Conclusions and Extensions

In the preceding sections we have argued that the Sparsity Problem is important, and one way to attack it has been explored. The key to our approach is that by assuming the Matching Property, the One Row Sparsity Problem can be solved (as shown in Section 3.2). The resulting One Row Algorithm is at the heart of the subsequent development.

The Parallel and Sequential Algorithms of Section 3.3 are not much more than clever ways of applying the One Row Algorithm to each row. Proving their correctness is not trivial, but some interesting theoretical results are obtained in return.

Section 3.4 shows that additional effort is required to bring the Sequential Algorithm to a point where it can be applied to real-life matrices. The practical algorithm described there seems to work reasonably well judging by the computational results in Section 3.5, but different and possibly better ways of implementing the algorithm are possible.

One possibility is a two-pass algorithm that separates the combinatorial and numerical parts of the algorithm, as follows. Theorem 3.3.1 implies that the parallel U_i^* induce an ordered decomposition, the SP Decomposition, on the rows of A (see Section 3.3). The SP Decomposition can be found by applying PA purely combinatorially to A and observing the sparsity pattern of T^*.

The proof of Theorem 3.3.2 shows that the effect of T^* on A is to transform "diagonal blocks" of A (with respect to some matching) into diagonal submatrices, and "subdiagonal blocks" (with respect to the same matching) into zero submatrices. The pattern of these diagonal and zero blocks is closely related to the sparsity pattern of T^* (see the example in the proof of Theorem 3.3.2). What is happening is that the fixed

matching picks out a square submatrix B of A. The matrix T^* is composed of various pieces of B^{-1}, the particular pieces being determined by the SP Decomposition of A. Thus $\hat{A} = T^*A$ is the result of performing only part of the Gauss-Jordan reduction of A that turns B into a diagonal matrix.

Since the SP Decomposition can be determined combinatorially, it can be known in advance which part of the Gauss-Jordan reduction of A to perform. Instead of letting a fixed, combinatorially-chosen matching determine the submatrix B at the start, a partial Gauss-Jordan elimination could be performed with dynamic, numerically-controlled pivot choices. The pivot choices would then implicitly determine the submatrix B, but only *after* the numerical operations. Such an algorithm would clearly be theoretically optimal since it eliminates as many non-zeros from A as PA does.

More importantly, a two-pass algorithm is also practically implementable. Since its operations are driven by numerical choices rather than by combinatorial choices, no difficulties would be encountered in applying it to matrices without (MP). It might also have an advantage in efficiency over SA.

To see how such an advantage would arise, note that the goal of Gauss-Jordan elimination is to reduce the partitioned matrix $(B \quad C)$ (where B is square and non-singular) to $(I \quad B^{-1}C)$. In the dense case, Gauss-Jordan elimination takes $O(n^3)$ operations. The Sequential Algorithm way to achieve the reduction is to solve for the multipliers that reduce each row separately, even though the multipliers are interrelated. Since one such solve uses $O(n^3)$ operations in the dense case, this would give an $O(n^4)$ algorithm overall. Similar savings might occur in the sparse case as well.

There are two difficulties with the two-pass algorithm that have prevented its implementation and comparison with SA. The first is that we prefer a more elegant way of computing the SP Decomposition than running PA combinatorially. The whole point of the SP Decomposition is that the U_i^* are themselves interrelated, and hence there should be a way to use the relationships to help compute the U_i^*. The ideal algorithm would globally develop the SP Decomposition, rather than generating it row by row as the combinatorial PA does. It would be esthetically pleasing if the combinatorial phase were to parallel the block (as opposed to row by row) nature of the numerical phase. However, such an algorithm is not yet at hand.

The second difficulty is that a practical implementation of the two-pass algorithm would require its own numerical subroutine to perform the partial Gauss-Jordan reduction. The existence of MA28 allowed implementation of SPARSER in a relatively short period of time. (Indeed, a private communication with Duff reveals that SPARSER is the only application of the capabilities for solving rectangular systems in MA28 of which he is aware.) Writing such a piece of software so that it is efficient, takes full advantage of sparsity and is numerically stable is a momentous undertaking.

There is another possible strategy which may improve the practicality of the algorithm. The sparsity of A is globally improved in the hope that on average the bases are then sparser. The objective of the Sparsity Problem implicitly assumes that any column is equally likely to appear in a basis. In some situations there is *a priori* knowledge about which columns are more likely to appear in a basis than other columns. Past experience or physical considerations of a model might lead to such knowledge. Alternatively, in an optimization with many iterations, the frequencies with which each column appears in a basis could be recorded in order to apply an algorithm that can take advantage of such information in the midst of the optimization. Such a strategy would allow the sparsity of A to be dynamically adjusted to reflect information about the columns that are most active during a long optimization run.

In either case, the problem of interest is the **Weighted Sparsity Problem (WSP)**. The WSP is the same as the regular Sparsity Problem except each column has a weight which represents an estimate of how likely that column is to be in a basis. The objective of WSP is to transform A into an $\hat{A}$ which has a minimum *weighted* number of non-zeros. WSP is another area for future research.

As mentioned at the end of Section 3.5, the computational results are only indicative, not conclusive. It would be very interesting to assemble a large collection of typical large-scale problems with which to test SPARSER, and to address the following questions. Are the reductions achieved by SPARSER in Table 3.5.2 typical? Is there any association between the form of an optimization problem and SPARSER's performance on it? (None appears obvious from Table 3.5.2, but perhaps there is a link with the source of the model or form of the matrix.) Are the conclusions drawn from Table 3.5.1 still valid on other matrices, particularly those with a higher ratio of "lucky gain" to "combinatorial gain?" Is there a stronger association between reduction in non-zeros and speed-up in optimization time than is evident in Table 3.5.3? Finally, and most

importantly, does SPARSER save more than it costs for large problems?

Bibliography

Bixby, R. and W. Cunningham (1983). Converting linear programs to network problems, to appear in *Math. of OR.*

Bollabás, B. (1978). *Extremal Graph Theory,* Academic Press, London and New York.

Bondy, J. A. and U. S. R. Murty (1976). *Graph Theory with Applications,* MacMillan, London.

Brent, R. P. (1973). Reducing the retrieval time of scatter storage techniques, *Comm. ACM,* **16**, pp. 105–109.

Coleman, T. F. and J. J. Moré (1981). Estimation of sparse Jacobian matrices and graph coloring problems, Report ANL-81-39, Argonne National Laboratory, Argonne, IL.

Coleman, T. F. and J. J. Moré (1982). Estimation of sparse Hessian matrices and graph coloring problems, Cornell University Department of Computer Science Report TR 82-535, Ithaca, NY.

Cottle, R. W. (1974). Manifestations of the Schur Complement, *Linear Algebra and its Applications,* **8**, pp. 182–211.

Curtis, A. R., M. J. D. Powell and J. K. Reid (1974). On the estimation of sparse Jacobian matrices, *Journal of the Institute of Mathematics and its Applications,* **13**, pp. 117–119.

Dahlquist, G. and Å. Björck (1974). *Numerical Methods,* Prentice-Hall, Englewood Cliffs, NJ.

Dantzig, G. B. (1963). *Linear Programming and Extensions,* Princeton University Press, Princeton, NJ.

Dantzig, G. B., B. Avi-Itzhak, T. J. Connolly, W. D. Winkler, et al., (1981). PILOT-1980 energy-economic model, volume 1, Electric Power Research Institute, Report EA-2090, Stanford, CA.

Dencker, P., K. Dürre and J. Heuft (1981). Optimization of parser tables for portable compilers, Universität Karlsruhe Interner Bericht Nr. 22/81, Karlsruhe.

Duff, I. S. (1977). MA28 — a set of FORTRAN subroutines for sparse unsymmetric linear equations, A.E.R.E. Harwell Report 8730.

Dulmage, A. L. and N. S. Mendelsohn (1963). Two algorithms for bipartite graphs, *J. SIAM,* **11**, pp. 183–194.

Dürre, K. and G. Fels (1980). Efficiency of sparse matrix storage techniques, in *Discrete Structures and Algorithms* (U. Pape ed.), München-Wien, pp. 209–221.

Ford, L. R. and D. R. Fulkerson (1962). *Flows in Networks,* Princeton University Press, Princeton, NJ.

Garey, M. R. and D. S. Johnson (1979). *Computers and Intractability,* Freeman, San Francisco, CA.

George, J. A. and F. G. Gustavson (1980) A new proof on permuting to block triangular form, IBM RC Report 8238, Yorktown Heights, NY.

Gill, P. E., W. Murray and M. H. Wright (1981). *Practical Optimization,* Academic Press, London and New York.

Golumbic, M. C. (1980). *Algorithmic Graph Theory and Perfect Graphs,* Academic Press, London and New York.

Grimmet, G. R. and C. J. H. McDiarmid (1975). On colouring random graphs, *Proceedings of the Cambridge Philosophical Society,* **77**, pp. 313–324.

Gustavson, F. G. (1973). Permuting matrices stored in sparse format, Disclosure Number 8-72-001, *IBM Technical Disclosure Bulletin,* **16**, 1, pp. 357–359.

Gustavson, F. G. (1976). Finding the block triangular form of a sparse matrix, in *Sparse Matrix Computations,* Academic Press, New York.

Hausmann, D. and B. Korte (1981). Algorithmic versus axiomatic definitions of matroids, *Math. Prog. Study,* **14**, pp. 98–111.

Hillier, F. S. and G. J. Lieberman (1974). *Operations Research,* second edition, Holden-Day, San Francisco, CA.

Hoffman, A. J. (1982). Personal communication.

Hoffman, A. J. and S. T. McCormick (1982). A fast algorithm that makes matrices optimally sparse, Stanford University Systems Optimization Laboratory Report 82-13, Stanford, CA (a revised version is to appear in the *Proceedings of the Silver Jubilee Conference on Combinatorics,* University of Waterloo, Waterloo, Ontario, (1983)).

Hopcroft, J. E. and R. M. Karp (1973). An $n^{5/2}$ algorithm for maximum matching in graphs, *SIAM J. Computing,* **2**, 4, pp. 225–231.

Iri, M. (1983). Structural theory for the combinatorial systems characterized by submodular functions, to appear in the *Proceedings of the Silver Jubilee Conference on Combinatorics,* University of Waterloo, Waterloo, Ontario.

Itai, A. and M. Rodeh (1978). Finding a minimum circuit in a graph, *SIAM J. Computing 7,* **4**, pp. 413–423.

Johnson, D. S. (1974). Worst case behavior of graph coloring algorithms, in *Proceedings of the 5th Southeastern Conference on Combinatorics, Graph Theory, and Computing,* Utilitas Mathematica, Winnipeg, Manitoba, pp. 513–527.

Karp, R. M. (1972). Reducibility among combinatorial problems, in *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), Plenum, New York, pp. 85–103.

Kaneko, I., M. Lawo and G. Thierauf (1982). On computational procedures for the force method, *Int. J. for Num. Meth. in Engr.,* **18**, pp. 1469–1495.

Knuth, D. E. (1973). *The Art of Computer Programming,* Volume 1, Second Edition, Addison-Wesley, Menlo Park, CA.

Lawler, E. L. (1976). *Combinatorial Optimization,* Holt, Rinehart and Winston, New York.

Markowitz, H. M. (1957). The elimination form of the inverse and its application to linear programming, *Management Science,* **3**, pp. 255–269.

McCormick, S. T. (1981). Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem, Stanford University Systems Optimization Laboratory Report SOL 81-22, Stanford, CA (a revised version appears in *Math. Prog.,* **26** (1983), pp. 153–171).

McCormick, S. T. (1983). A combinatorial approach to some sparse matrix problems, Ph. D. Thesis, Stanford University, Stanford, CA.

Murtagh, B. A. (1981). *Advanced Linear Programming and Practice,* McGraw-Hill, New York.

Newsam, G. N. and J. D. Ramsdell (1982). Estimation of sparse Jacobian matrices, Harvard University Division of Applied Sciences Report TR-17-81, Cambridge, MA.

Papadimitriou, C. H. and K. Stieglitz (1982). *Combinatorial Optimization: Algorithms and Complexity,* Prentice Hall, Englewood Cliffs, NJ.

Peters, G. and J. H. Wilkinson (1970). The least squares problem and pseudo-inverses, *Computer J.,* **13**, pp. 309–316.

Powell, M. J. D. and P. L. Toint (1979). On the estimation of sparse Hessian matrices, *SIAM Journal on Numerical Analysis,* **16**, pp. 1060–1074.

Ryser, H. J. (1963). *Combinatorial Mathematics,* MAA Camus Mathematical Monograph Number 14, Providence, RI.

Saks, M. and J. Kahn (1983). Personal communication.

Saunders, M. A. (1977). MINOS system manual, Stanford Systems Optimization Laboratory Report SOL 77-31, Stanford, CA.

Stockmeyer, L. J. (1982). Personal communication.

Thapa, M. N. (1980). Optimization of unconstrained functions with sparse Hessian matrices, Ph. D. Thesis, Stanford University, Stanford, CA.

Welsh, D. (1976). *Matroid Theory,* Academic Press, London and New York.

Widgerson, A. (1982). A new appproximate graph coloring algorithm, *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing.*

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: SOL 83-5

2. GOVT ACCESSION NO.:

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE *(and Subtitle)*: A COMBINATORIAL APPROACH TO SOME SPARSE MATRIX PROBLEMS

5. TYPE OF REPORT & PERIOD COVERED: Technical Report

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): S. Thomas McCormick

8. CONTRACT OR GRANT NUMBER(s): N00014-75-C-0267
DAAG29-81-K-0156

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Department of Operations Research - SOL
Stanford University
Stanford, CA 94305

Office of Naval Research - Dept. of the Navy
800 N. Quincy Street
Arlington, VA 22217

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NR-047-143

12. REPORT DATE: June 1983

13. NUMBER OF PAGES: 69 PP

14. MONITORING AGENCY NAME & ADDRESS *(If different from Controlling Office)*
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709

15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT *(of this Report)*

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

sparse Hessian approximation
graph coloring heuristics
sparse matrices
bipartite matching
computational complexity
NP-complete
direct methods
elimination methods
linear constraints

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

SEE ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

SOL 83-5, "A COMBINATORIAL APPROACH TO SOME SPARSE MATRIX PROBLEMS,"
by S. Thomas McCormick

This dissertation considers two combinatorial problems arising in large-scale, sparse optimization. The first is the problem of approximating the Hessian matrix of a smooth, non-linear function by finite differencing, where the object is to minimize the required number of gradient evaluations. The second is to find as sparse a representation as possible of a given set of linear constraints.

For the first problem, it has recently been realized that when the Hessian has a fixed, known sparsity pattern, a considerable reduction in gradient evaluations can often be achieved by a suitable choice of difference directions. This dissertation advances a way of classifying the various methods that have been proposed for choosing difference directions, and shows that finding an optimally small set of directions for any of the four subvarieties of the Direct Methods is NP-Complete. The complexity results are obtained by showing that finding optimal sets of difference directions is equivalent to related graph coloring problems. Some results for more general methods are reported that yield good lower bounds on the minimum number of gradient evaluations needed to approximate many Hessians.

The second problem has been shown to be NP-Complete. By adopting a fairly mild non-degeneracy assumption we are able to derive a low-order polynomial algorithm which reduces given constraints into an optimally sparse equivalent set of constraints. This algorithm is based on bipartite matching theory, and it induces a partial order on the rows of the matrix which is related to Dulmage-Mendelsohn decomposition. The proof that the algorithm is correct yields a performance guarantee when the algorithm is applied to real data, and several modifications that improve its running time are discussed. Some computational experience is presented which indicates that the algorithm may be practically useful as a preprocessor for linearly constrained optimization. We also discuss the relationship of this research to finding optimally sparse null-space bases, and to the complexity of matroid oracles.