Fast Numerical Determination of Symmetric Sparsity Patterns

Richard G. Carter

October 1992

Argonne Laboratory Preprint MCS-P326-0992

Fast Numerical Determination of Symmetric Sparsity Patterns [*]

Richard G. Carter [†]

Abstract

We consider a function $g : \Re^n \to \Re^n$ for which the Jacobian is symmetric and sparse. Such functions often arise, for instance, in numerical optimization, where g is the gradient of some objective function f so that the Jacobian of g is the Hessian of f. In many such applications one can generate extremely efficient algorithms by taking advantage of the sparsity structure of the problem if this pattern is known a priori. Unfortunately, determining such sparsity structures by hand is often difficult and prone to error. If one suspects a mistake has been made, or if g is a "black box" so that the true structure is completely unknown, one often has no alternative but to compute the entire matrix by finite differences — a prohibitively expensive task for large problems.

We show that it is possible to numerically determine symmetric sparsity patterns using a relatively small number of g evaluations. Numerical results are shown for n up to 100,000 in which all nonzeros in the Jacobian are correctly identified in about one-hundredth of the time required to estimate the sparsity structure by a full finite difference calculation. When a good initial guess for the sparsity structure is available, numerical results are presented for n up to 500,000, in which all missing nonzeros are correctly located almost five-thousand times faster than would be possible with a full finite difference calculation.

1 Introduction

Consider a function $g : \Re^n \to \Re^n$ for which the Jacobian $\nabla g^T : \Re^n \to \Re^{n\times n}$ is symmetric and sparse. Such systems arise, for instance, in numerical optimization, where $g \equiv \nabla f$ for some objective function f so that the Jacobian of g is the Hessian matrix $\nabla^2 f$. These systems are particularly common when the variables in the optimization problem correspond to mesh points in a discretization. For instance, Figure 1 represents the sparsity pattern arising from the optimal design of a composite material when regular triangular finite elements are used in the model discretization [1], while Figure 2 represents the more complex sparsity pattern arising from the solution of the Ginzburg-Landau equations in two dimensions for superconducting materials [1],[7].

In many applications one can generate extremely efficient algorithms by taking advantage of the sparsity structure of ∇g if this structure is known. It is well known that if the columns of ∇g can be partitioned into k groups such that within each group no two columns share a nonzero element in the same row, then the entire matrix can be approximated using only k evaluations of g. Moreover, the matrix can be determined *exactly* using values for the directional derivative of g in k specific directions, if such directional derivatives can be computed. Since k is typically much less than n unless the sparsity structure is pathological, ∇g can be determined very inexpensively *provided* the sparsity structure is known and the columns can be efficiently partitioned into groups. Determining such a partition given a specified sparsity structure is a graph coloring problem and

[*]This research was supported by the Army Research Office under grant DAALO3-89-C-0038, and in part by the Applied Mathematical Sciences Subprogram of the Office of Energy Research of the US Department of Energy under contract W-31-109-Eng-38.

[†]Army High Performance Computing Research Center, Institute of Technology, University of Minnesota, 1100 Washington Avenue South, Minneapolis, Minnesota 55415

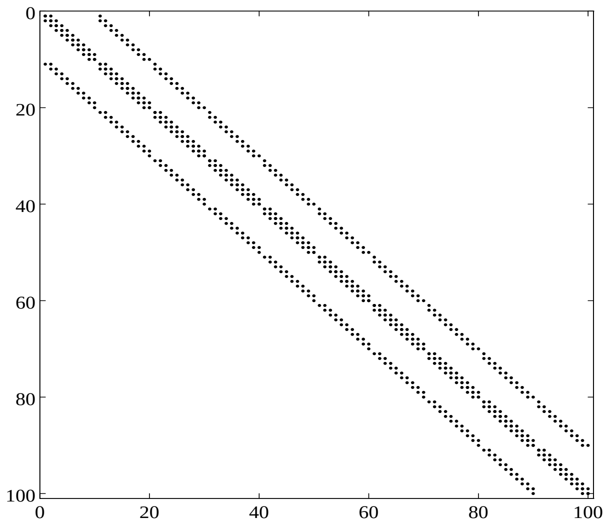

Figure 1: Symmetric Sparsity Structure for Optimal Design Problem

has been well analyzed [6], [4]. Efficient software for computing a nearly optimal partition and using it for determining ∇g is widely available in the public domain [3]. For the nonsymmetric case we refer to this procedure as the CPR method. (Specialized methods exist for reducing work even further in the symmetric case (see, for instance, [8], [5], and [2]), but for reasons that will become clear later, we use the CPR method as the foundation of our algorithm.)

Unfortunately, correctly specifying the sparsity structure by hand calculation is often difficult, tedious, and prone to error. Including extra elements in the structure that later turn out to be zero is no particular problem, but leaving elements out of the structure specification will cause the CPR procedure to return erroneous results. If one suspects such a mistake has been made, or if g is a "black box" so that one has no idea of the true structure, there is often no alternative but to compute the entire matrix by finite differences (using n evaluations of g) in order to find the location of the "missing" elements — a prohibitively expensive task for large problems. We refer to this as the FFD (full finite difference) approach.

Let $\mathcal{S}_*$ denote the true sparsity structure of ∇g, and let $\mathcal{S}_0$ denote an erroneous initial guess to $\mathcal{S}_*$. We show how to numerically determine the true sparsity pattern $\mathcal{S}_*$ using $\mathcal{S}_0$ and a relatively small number of g evaluations. The basic idea of the method is based on the crucial observation that symmetric pairs of missing elements (i.e., elements of $\mathcal{S}_*$ that are not in $\mathcal{S}_0$) will cause *nonsymmetric* perturbations in the output of the CPR algorithm. These perturbations can be located and used to reconstruct possible source locations for elements not included $\mathcal{S}_0$. We emphasize that this technique is a numerical procedure rather than a symbolic one; hence, we refer to our method as the NSSD (numerical sparsity structure determination) algorithm.

In some applications, ∇g may be technically dense yet sparse for practical purposes, in the sense that almost all the elements are of insignificant size. For instance, one test problem we consider has $\nabla g = A + B$, where the $5 * n$ elements of A all have magnitudes between 1 and 2 and have locations randomly distributed as in Figure 3, while B has $o(n^{3/2})$ elements of size 10^{-6} distributed[*] as in

[*]Specifically, the rows of nonzeros in the lower triangle of B are spaced $\sqrt{n}/2$ rows apart.

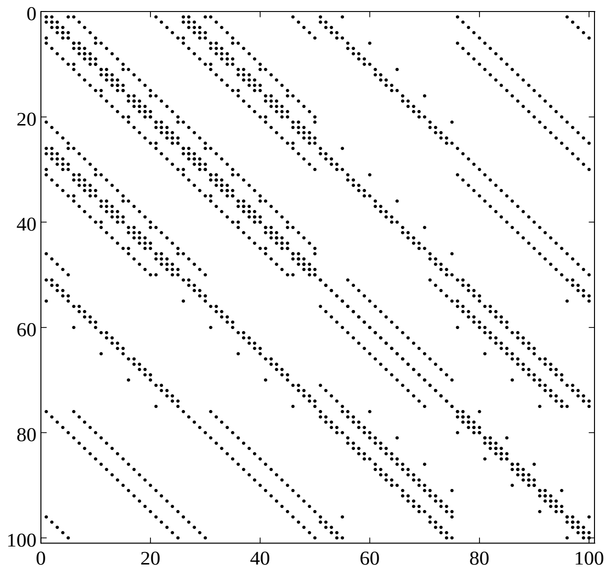

Figure 2: Symmetric Sparsity Structure for 2-D Ginzburg-Landau Equations

Figure 4. An important feature of our algorithm is the ability to "filter out" small elements by appropriate use of a number of tests with user-adjustable tolerances.

Although the basic idea behind our method is extremely simple, some of the details of the algorithm are somewhat involved. To motivate the reader, we preview some of our numerical results in the following tables. To put the computational times in perspective, we also compare the times required for our algorithm with the amount of time it would have taken to determine the structure by full finite differences. The test problem for the first table uses $\nabla g = A + B$, where A and B are described above. No initial guess for the structure was provided, and tolerances were set to ignore the elements of B. In each case all nonzero entries of A were correctly located.

Table 1. Performance of NSSD algorithm compared with full finite differences for $\nabla g = A + B$. No initial guess for the pattern provided by user.

n	nnz	Number of g evals	g evaluation time (sec)	Other time (sec)	Total NSSD time (sec)	Estimated time for a full finite difference Hessian to get structure ($n\times$ cost(g eval)) (sec)
2,000	10,008	187	19.5	34.6	54.1	220
5,000	24,994	264	114	148	262	2,475
10,000	50,000	238	335	293	628	14,650
20,000	100,015	318	1,350	2,760	4,110	82,200
50,000	250,015	346	5,830	6,300	12,130	816,251
100,000	500,021	477	22,100	32,500	54,600	4,686,499

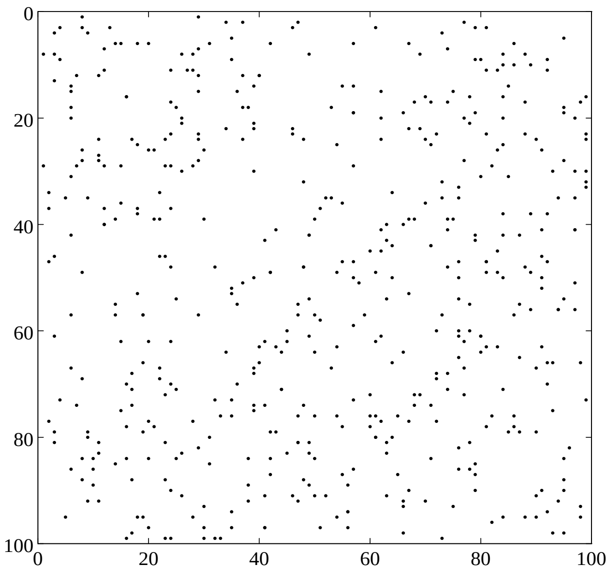

Figure 3: Symmetric Sparsity Structure for Random Pattern A

The quantity nnz denotes the number of elements in the true sparsity pattern. Note that for the $n = 100,000$ case, NSSD is almost 100 times faster than full finite differences.

If a good initial guess for the true structure is provided, these times decrease even further. Moreover, because of lower storage requirements we can solve larger problems. In the next example (Table 2) the initial guess for the structure was set to be the true structure of A less 1000 entries. Again, tolerances were set to ignore the elements of B. In each case all 1000 missing entries of A were correctly located. Note that for the $n = 500,000$ case, NSSD is almost 5000 times faster than full finite differences.

Table 2: Performance of NSSD algorithm † and comparison to FFD, with 1000 elements missing from the initial guess.

n	nnz	Number of g evals	g evaluation time (sec)	Other time (sec)	Total NSSD time	Estimated time for a full finite difference Hessian to get structure ($n\times$ cost(g eval))
2,000	10,008	121	12.7	7.76	20.5 sec	220.0 sec
5,000	24,994	102	50.8	24.7	75.5 sec	2,475.0 sec
10,000	50,000	117	172	48.6	3.7 min	4.1 hr
20,000	100,015	111	453	75.9	8.8 min	22 hr
50,000	250,015	245	4,000	388	1.2 hr	9 days
100,000	500,021	176	8,270	671	2.5 hr	54 days
200,000	999,994	341	45,400	3,200	13.5 hr	307 days
500,000	2,499,984	103	51,000	1,990	14.8 hr	7.9 years

†Some tolerances were hand-tuned for the $n = 500,000$ case.

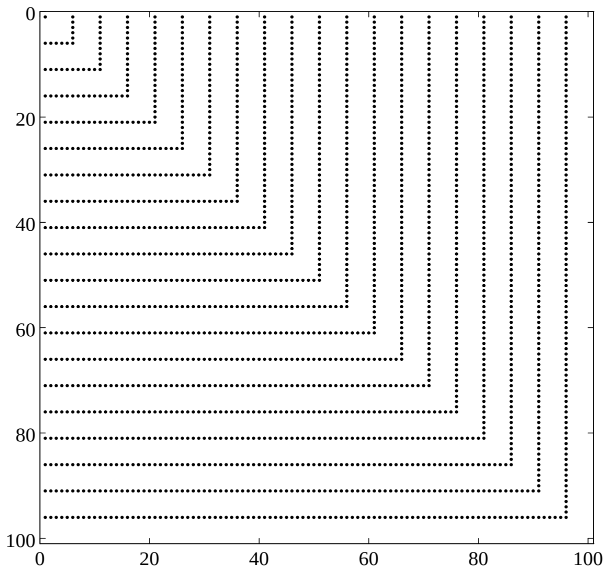

Figure 4: Symmetric Sparsity Structure for B

Of course, for problems with very inexpensive g evaluations, the contrast is not so great, but the new method is still significantly faster than finite differences. If we use the slightly different example $\nabla g = A$ rather than $\nabla g = A + B$, then g can be evaluated with about 10*n operations, which is about as inexpensive as one can get. For such cases, algorithm overhead dominates the expense of computing the g values. With no initial guess provided, the algorithm correctly located all nonzeros in the times shown in Table 3.

Table 3. Performance of NSSD algorithm and comparison to FFD, for $\nabla g = A$ with no initial guess provided by user.

n	Number of g evals	g eval time (sec)	Other time (sec)	Estimated time for a full finite difference (sec)
2,000	175	2.65	22.0	29.2
5,000	211	9.63	102	225
10,000	223	24.4	314	1,098
20,000	296	70.6	1,450	4,764
50,000	300	189	5,980	31,240

For the "real" examples such as the composite optimal design problem and the 2-D Ginzburg-Landau problem, the NSSD algorithm is typically 10 to 100 times faster than full finite differences for problems in the n=2,000 to n=40,000 range.

These results will be discussed in greater detail in §6. We also note that if additional information is available in the form of an analytic expression for the diagonal of ∇g, NSSD performs even better.

The NSSD algorithm is best understood if it is first presented in simplest form and then modified in stages until the most general form of the algorithm is presented. In §2 we present notation and discuss the effects of nonempty $\mathcal{S}_* - \mathcal{S}_0$ on the CPR method. In §3 we present an algorithm that can quickly identify and locate a *small* number of missing elements, *provided* an analytic expression for the diagonal of ∇g is available. In §4 we show how this algorithm can be modified into a *multilevel* approach which can successfully handle extremely poor initial guesses $\mathcal{S}_0$. In §5 we show how the introduction of a voting scheme between the levels allows us to drop the restrictive assumption that an analytic expression for the diagonal of ∇g is available. In §6 we discuss the actual implementation of the algorithm, and present further numerical results. In §7 we summarize our results and suggest areas for future research.

2 Determination of Sparse Matrices by the CPR Algorithm

Notation. Script letters such as $\mathcal{S}$, $\mathcal{P}$, $\mathcal{R}$, and $\mathcal{F}$ refer to sets of indices of matrix elements, typically pointing to collections of elements within ∇g. These sets are also referred to as *patterns.* $\mathcal{P}^T$ denotes the transpose of a sparsity pattern $\mathcal{P}$. The quantity $nnz(\mathcal{S})$ denotes the number of elements in an index set $\mathcal{S}$. $\mathcal{S}_*$ is the pattern of the nonzero elements in ∇g; that is, $\mathcal{S}_*$ is the true sparsity structure. $\mathcal{S}_0$ is a symmetric initial guess for $\mathcal{S}_*$. $\mathcal{I}$ is the set of indices pointing to the diagonal of ∇g. $\mathcal{C}$ denotes a set of column indices.

We often denote the true value of ∇g by H_*, and the output of the CPR algorithm (using some pattern $\mathcal{S}$) by H. The individual elements of H are denoted by $h_{i,j}$ while columns of H are denoted by h_i; the distinction between these usages will be clear from context. Finally, we often refer to a subset of the columns of H by the notation $H_{\mathcal{C}}$.

The CPR algorithm. Consider the equation

$$H_* \, d = y, \tag{1}$$

where $d, y \in \Re^n$. Let $h_1^*, h_2^*, ..., h_n^*$ be the columns of H_*, and let $H_{\mathcal{C}}^* = \{h_j^* : j \in \mathcal{C}\}$ be a subset of the columns such that no two columns in $H_{\mathcal{C}}^*$ have a nonzero in the same row position. Curtis, Powell, and Reid [6] observed in 1974 that if d has components $d_j = \delta_j \neq 0$ if $j \in \mathcal{C}$ and $d_j = 0$ otherwise, then d and y uniquely determine $H_{\mathcal{C}}^*$. This fact can be established by noting

$$H_* d = \sum_{j \in \mathcal{C}} \delta_j h_j^*. \tag{2}$$

Since no pair of columns in $H_{\mathcal{C}}^*$ has a nonzero in the same row, we have

$$\delta_j h_{i,j}^* = y_i \tag{3}$$

for each nonzero $h_{i,j}^*$, $j \in \mathcal{C}$; hence, d and y uniquely determine $H_{\mathcal{C}}^*$.

The left-hand side of (1) is just the directional derivative of g at the point x in the direction d. Until recently the most common way to determine y was to approximate this directional derivative using forward or central differences:

$$y = g(x+d) - g(x) + o(\|d\|) \quad \text{or} \quad y = \frac{1}{2}(g(x+d) - g(x-d)) + o(\|d\|^2) \tag{4}$$

where each nonzero component of d is appropriately small. An alternative approach which has recently become very attractive is to use an automatic differentiation package such as ADIFOR to generate code for computing the expression $\nabla g^T d$ exactly for the direction d we have selected. For other applications, it is sometimes possible to write an analytic expression for the product $\nabla g^T d$ with much less human effort than is involved in deriving a complete expression for ∇g. In either of these alternatives, $H_{\mathcal{C}}^*$ will be reconstructed exactly (except for roundoff) for any (nonzero) choice of δ_i. In describing our algorithm in §2 through §5, we assume for simplicity that y values are exact, but in §6 and in our numerical examples in §1, we have used forward differences.

If the columns of H_* can be partitioned into k groups such that within each group $\mathcal{C}_j$ no two columns share a nonzero row element, Equation (1) can be expressed in matrix form as

$$H_* D = Y, \tag{5}$$

where $D, Y \in \Re^{n \times k}$ and the index sets $\{\mathcal{C}_j\}, \mathcal{S}_*$ together determine $H = H_*$ uniquely by the formula

$$h_{i,j} = y_{i,k}/\delta_j \tag{6}$$

for each (i, j) inside the pattern $\mathcal{S}_*$ with k the group number of column j, and $h_{i,j} = 0$ for indices outside the pattern.

Coleman and Moré [4] have established that the task of efficiently partitioning the columns is a graph coloring problem and have published software [3] that computes a nearly optimal partition (coloring) for a given pattern. A description of their coloring algorithm is beyond the scope of this paper, but we remark that if a pattern has a maximum of k nonzeros per row, (a) the best we can expect is a partitioning into k groups, and (b) the Coleman-Garbow-Moré code [3] typically does almost this well. The reader is cautioned that pathological examples do exist for which the number of groups is as large as n, but such cases are rare in practice.

Example 1. The best way to understand the CPR algorithm is to consider an actual example. Suppose

$$H_* = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

The columns of this matrix can be easily partitioned into two groups. For this example, we select group $\mathcal{C}_1$ to consist of columns 1, 2 ,5, and 8, while group $\mathcal{C}_2$ consists of columns 3, 4, 6, and 7. The patterns for the two partitions can be represented by

$$\mathcal{S}_0 = \begin{bmatrix} \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \end{bmatrix}. \tag{8}$$

(The symbols $\heartsuit$ and $\clubsuit$ are used to distinguish between columns belonging to different groups.) Equation (5) is then

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_1 & 0 \\ \delta_2 & 0 \\ 0 & \delta_3 \\ 0 & \delta_4 \\ \delta_5 & 0 \\ 0 & \delta_6 \\ 0 & \delta_7 \\ \delta_8 & 0 \end{bmatrix} = \begin{bmatrix} \delta_1 & \delta_3 \\ \delta_2 & 0 \\ \delta_1 & \delta_3 \\ \delta_5 & \delta_4 \\ \delta_5 & \delta_4 \\ 0 & \delta_6 \\ 0 & \delta_7 \\ \delta_8 & 0 \end{bmatrix}. \tag{9}$$

It is easy to verify that expanding D and Y into a dense matrix format using Equation (6) and the pattern $\mathcal{S}_*$ yields the original matrix H_*.

The CPR algorithm with an incorrect initial pattern. Suppose elements (1,4) and (4,1) are actually nonzero. That is, we have erroneously specified the pattern to be $\mathcal{S}_0$ rather than the true structure $\mathcal{S}_*$, where

$$\mathcal{S}_0 = \begin{bmatrix} \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \end{bmatrix} \quad \text{and} \quad \mathcal{S}_* = \begin{bmatrix} \heartsuit & \cdot & \clubsuit & \clubsuit & \cdot & \cdot & \cdot & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \end{bmatrix}. \tag{10}$$

Although the columns of $\mathcal{S}_0$ are partitioned "correctly," notice that columns 3 and 4 overlap in row 1 of $\mathcal{S}_*$ even though the two columns are in the same group, and columns 1 and 5 similarly overlap in row 4 even though they are in the same group.

Let us denote the (1,4) element by α and the (4,1) element by β so that we may distinguish between them. If we apply the CPR algorithm using the erroneous structure $\mathcal{S}_0$, Equation (5) becomes

$$\begin{bmatrix} 1 & 0 & 1 & \alpha & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \beta & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_1 & 0 \\ \delta_2 & 0 \\ 0 & \delta_3 \\ 0 & \delta_4 \\ \delta_5 & 0 \\ 0 & \delta_6 \\ 0 & \delta_7 \\ \delta_8 & 0 \end{bmatrix} = \begin{bmatrix} \delta_1 & \delta_3 + \delta_4\alpha \\ \delta_2 & 0 \\ \delta_1 & \delta_3 \\ \delta_1\beta + \delta_5 & \delta_4 \\ \delta_5 & \delta_4 \\ 0 & \delta_6 \\ 0 & \delta_7 \\ \delta_8 & 0 \end{bmatrix}. \tag{11}$$

Using the sparsity template $\mathcal{S}_0$ and Equation (6) to expand D and Y out into a full matrix yields

$$H = \begin{bmatrix} 1 & 0 & 1 + \alpha\frac{\delta_4}{\delta_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 + \beta\frac{\delta_1}{\delta_5} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{12}$$

Now, examine Equations (10) — (12). Clearly, the overlap at row 4 in columns 1 and 5 manifested within the $\mathcal{S}_0$ pattern as a perturbation to element $(4,5)$, while the overlap at row 1 in columns 3 and 4 manifested within the $\mathcal{S}_0$ pattern as a perturbation to element $(1,3)$. *Note that these perturbations are not symmetric* even though by hypothesis $H_* = H_*^T$.

Example 2. Another possible effect of missing elements is demonstrated by the following example with "missing" elements at locations $(1,7)$ and $(7,1)$:

$$\mathcal{S}_0 = \begin{bmatrix} \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \end{bmatrix} \quad \text{and} \quad \mathcal{S}_* = \begin{bmatrix} \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \end{bmatrix}. \tag{13}$$

Notice that columns 3 and 7 overlap in row 1 of the true sparsity pattern, but the addition of the $(7,1)$ element to column 1 does *not* cause column 1 to overlap with any other column in group 1. Performing the same calculations as in the preceding example, we get

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & \alpha & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ \beta & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \delta_1 & 0 \\ \delta_2 & 0 \\ 0 & \delta_3 \\ 0 & \delta_4 \\ \delta_5 & 0 \\ 0 & \delta_6 \\ 0 & \delta_7 \\ \delta_8 & 0 \end{bmatrix} = \begin{bmatrix} \delta_1 & \delta_3 + \delta_7\alpha \\ \delta_2 & 0 \\ \delta_1 & \delta_3 \\ \delta_5 & \delta_4 \\ \delta_5 & \delta_4 \\ 0 & \delta_6 \\ \delta_1\beta & \delta_7 \\ \delta_8 & 0 \end{bmatrix}. \tag{14}$$

Using the sparsity template $\mathcal{S}_0$ to expand D and Y out into a full matrix yields

$$H = \begin{bmatrix} 1 & 0 & 1+\alpha\frac{\delta_7}{\delta_3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{15}$$

First notice that the nonzero in element $(1,7)$ of H_* caused a perturbation of element $(1,3)$ in H — the same H element perturbed in the preceding example by a nonzero in $(1,4)$.

Second, carefully compare the right-hand sides of Equations (9) and (14). In this example, β at $(7,1)$ did *not* appear as a perturbation of any element of H, but instead appeared in the matrix Y at a location that is not used in further computation. That is, if our pattern were correct and $\beta = 0$, then $y_{7,1}$ would have been zero automatically and would not be needed to compute H. Using the incorrect pattern $\mathcal{S}_0$, the CPR algorithm never accesses $y_{7,1}$ in computing H.

In general, we have that an element in $\mathcal{S}_*$ but not in $\mathcal{S}_0$ will affect our computation by perturbing an element of H in the same row and in the same group. If no such element exists, then the perturbation will appear in the right-hand side Y in the same row and in the column corresponding to the group, a location that would normally contain zero if the pattern were correct. Formally stated, this becomes the following.

Proposition 1: Suppose $h^*_{i,j}$ is a nonzero element in the symmetric matrix H_* but (i,j) is not included in $\mathcal{S}_0$. Let the matrix H be generated by the CPR algorithm using pattern $\mathcal{S}_0$, and let m be the number of the group to which column j belongs. Then H will be perturbed in element (i,k), where (i,k) is a member of $\mathcal{S}_0$ and column k is in the group m. If such a k does not exist, then the (i,m) element of Y will be perturbed.

3 Detection and Localization of a Limited Number of Missing Nonzero Elements

Since we now know the effect on the CPR algorithm if one misspecifies the pattern, it is only natural to ask whether one can use these effects to determine the location of missing pattern elements. Making a few definitions and inverting the reasoning of Proposition 1 yields the following.

Definition 1: Let the matrix H and the coloring $\{\mathcal{C}_j\}$ be the result of applying the CPR method with pattern $\mathcal{S}_0 \neq \mathcal{S}_*$. If $h_{i,j} \neq h^*_{i,j}$, then $h(i,j)$ is said to be an H *flaw*. The set of all H flaws, or an approximation to this set, is denoted $\mathcal{F}^h$.

Definition 2: Let Y be the right-hand side obtained when applying the CPR method with pattern $\mathcal{S}_0 \neq \mathcal{S}_*$. If $y_{i,m} \neq 0$, but $\mathcal{S}_0$ does not contain an element (i,j) with $j \in \mathcal{C}_m$, then $y(i,m)$ is said to be a Y *flaw*. The set of all Y flaws, or an approximation to this set, is denoted $\mathcal{F}^y$.

Proposition 2: Let the matrix H and the coloring $\{\mathcal{C}_j\}$ be the result of applying the CPR method with pattern $\mathcal{S}_0 \neq \mathcal{S}_*$. If $h_{i,j}$ is an H flaw and column j belongs to group m, then the pattern $\mathcal{S}_0$ is missing one or more elements in row i. Moreover, the set of possible locations for these missing elements is $\{(i,k) : k \in \mathcal{C}_m\}$.

Proposition 3: If $y_{i,m}$ is a Y flaw, then the pattern $\mathcal{S}_0$ is missing one or more elements in row i. Moreover, the set of possible locations for these missing elements is $\{(i,k) : k \in \mathcal{C}_m\}$.

Definition 3: $\mathcal{P}(\mathcal{F}^h)$ denotes the pattern of possible locations specified by Proposition 2. $\mathcal{P}(\mathcal{F}^y)$ denotes the pattern of possible locations specified by Proposition 3. $\mathcal{P}(\mathcal{F})$ denotes $\mathcal{P}(\mathcal{F}^h) \cup \mathcal{P}(\mathcal{F}^y)$.

Example 1, continued. Let us return to Example 1 (Equation (10)) and apply the above definitions. For the sake of argument, suppose that we have somehow determined the correct location of the flaws in H:

$$\mathcal{F}^h = \begin{bmatrix} \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathcal{F}^y = \emptyset. \tag{16}$$

The pattern of possibilities is

$$\mathcal{P}(\mathcal{F}) = \mathcal{P}(\mathcal{F}^h) = \begin{bmatrix} \cdot & \cdot & \cdot & \clubsuit & \cdot & \clubsuit & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \tag{17}$$

Recall, however, that H_* is symmetric. Eliminating nonsymmetric entries among the possibilities yields

$$(\mathcal{P}(\mathcal{F}))^T \cap \mathcal{P}(\mathcal{F}) = \begin{bmatrix} \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \tag{18}$$

This is precisely the set $\mathcal{S}_* - \mathcal{S}_0$: we have correctly identified the missing elements in the pattern.

Example 2, continued. In this example our flaws are

$$\mathcal{F}^h = \begin{bmatrix} \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathcal{F}^y = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \\ \heartsuit & \cdot \\ \cdot & \cdot \end{bmatrix}, \tag{19}$$

so our pattern of possibilities is

$$\mathcal{P}(\mathcal{F}) = \mathcal{P}(\mathcal{F}^h) \cup \mathcal{P}(\mathcal{F}^y) = \begin{bmatrix} \cdot & \cdot & \cdot & \clubsuit & \cdot & \clubsuit & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \quad (20)$$

Eliminating nonsymmetric entries among the possibilities yields

$$(\mathcal{P}(\mathcal{F}))^T \cap \mathcal{P}(\mathcal{F}) = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}. \quad (21)$$

Again, this is precisely the set $\mathcal{S}_* - \mathcal{S}_0$.

Example 3. Although in the preceding two examples we were able to uniquely determine the missing elements, nothing precludes elements of $\mathcal{P}$ and $\mathcal{P}^T$ from matching *by accident.* If this situation occurs, we will obtain spurious possibilities as well as the pattern locations for which we are looking. Consider the following example:

$$\mathcal{S}_0 = \begin{bmatrix} \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit \end{bmatrix}, \quad (22)$$

where

$$\mathcal{S}_* = \begin{bmatrix}
\heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\
\heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot \\
\cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \spadesuit \\
\heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit & \heartsuit & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \diamondsuit & \cdot \\
\cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \spadesuit \\
\cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \spadesuit
\end{bmatrix}. \tag{23}$$

We have partitioned $\mathcal{S}_0$ into three groups. The true stencil $\mathcal{S}_*$ contains twelve elements not found in $\mathcal{S}_0$. After determining H using the CPR algorithm, the flaws are

$$\mathcal{F}^H = \begin{bmatrix}
\heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit
\end{bmatrix} \quad \text{and} \quad \mathcal{F}^y = \emptyset. \tag{24}$$

Then the stencil of possibilities $\mathcal{P}$ is

$$\mathcal{P} = \begin{bmatrix}
\cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot \\
\cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot \\
\heartsuit & \cdot & \cdot & \cdot & \cdot & \spadesuit & \heartsuit & \cdot & \spadesuit & \heartsuit & \cdot & \spadesuit \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \diamondsuit & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit \\
\heartsuit & \diamondsuit & \cdot & \heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \heartsuit & \diamondsuit & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit \\
\cdot & \diamondsuit & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot
\end{bmatrix}. \tag{25}$$

Eliminating nonsymmetric elements yields

$$\mathcal{P}^T \cap \mathcal{P} = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \heartsuit & \cdot & \spadesuit \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit \\
\heartsuit & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \spadesuit \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \diamondsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \spadesuit & \cdot & \cdot & \cdot
\end{bmatrix}. \tag{26}$$

This pattern contains the 12 missing elements in $\mathcal{S}_* - \mathcal{S}_0$, along with 8 spurious entries.

Determination of flaw locations. At this point we make the assumption that an analytic expression for the diagonal of H is available. (This assumption is indeed restrictive and will be dropped in later sections, but it simplifies our presentation.) Consider the following procedure for determining the flaw locations.

Procedure 1: Computation of flaw locations

1. **Given** the true values of h^*_{ii}, $i = 1, 2, ..., n$, and
2. **Given** the matrix H and the coloring $\{\mathcal{C}_j\}$ computed by the CPR algorithm using a symmetric trial pattern $\mathcal{S}$, and
3. **Given** tolerances $\epsilon_h, \epsilon_y > 0$.
4. **Set** $\mathcal{F}^h = \mathcal{F}^y = \emptyset$.
5. **For all** $(i, j) \in \mathcal{S}$ with $j \geq i$ **Do**:
   - **If** $i = j$ and
     $$|h_{i,i} - h^*_{i,i}| > \epsilon_h, \tag{27}$$
     then add (i, i) to the set $\mathcal{F}^h$.
   - **If** $i \neq j$ and
     $$|h_{i,j} - h_{j,i}| > \epsilon_h, \tag{28}$$
     then add (i, j) and (j, i) to the set $\mathcal{F}^h$.
6. **For all** $y_{i,j}$ **Do**:
   - **If** $\mathcal{S}$ contains no element (i, k) with $k \in \mathcal{C}_j$ and
     $$|y_{i,j}| > \epsilon_y \frac{1}{n} \sum_{k=1}^{n} |\delta_k| \tag{29}$$
     then add (i, j) to the set $\mathcal{F}^y$.
7. **Exit** procedure.

We remark that the computation of Y flaws can be easily performed during the CPR computation of H at almost no additional expense. Determination of H flaws is also extremely inexpensive computationally.

Rather than the absolute tests used above, we could also use relative tests for detecting flaws. Specifically, we could replace Equations (27), (28), and (29) with the tests

$$|h_{i,i} - h^*_{i,i}| > \epsilon_h \; max(|h_{i,i}|, |h^*_{i,i}|), \tag{30}$$

$$|h_{i,j} - h_{j,i}| > \epsilon_h \; max(|h_{i,j}|, |h_{j,i}|), \tag{31}$$

$$|y_{i,j}| > \epsilon_y \frac{1}{n} \sum_{k=1}^{n} |y_{k,j}|. \tag{32}$$

The absolute versions of the tests are preferable when one knows the approximate size of small elements to be filtered from the structure.

Procedure 1 will always tag both (i, j) and (j, i) as flawed if $h_{i,j}$ differs significantly from $h_{j,i}$. Essentially, we are saying that if we do not know which of the two elements is flawed, we will consider both to be. How does this affect our computations? Return once more to Example 1, and consider what happens if Procedure 1 is used to compute the location of the flaws.

$$\mathcal{F}^h = \begin{bmatrix} \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathcal{F}^y = \emptyset \tag{33}$$

$$\mathcal{P}(\mathcal{F}) = \begin{bmatrix} \cdot & \cdot & \cdot & \clubsuit & \cdot & \clubsuit & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \heartsuit & \cdot & \cdot & \heartsuit & \cdot & \cdot & \heartsuit \\ \heartsuit & \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \heartsuit \\ \cdot & \cdot & \clubsuit & \cdot & \cdot & \clubsuit & \clubsuit & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad \mathcal{P}(\mathcal{F})^T \cap \mathcal{P}(\mathcal{F}) = \begin{bmatrix} \cdot & \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \heartsuit & \cdot & \cdot & \cdot \\ \heartsuit & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \clubsuit & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix} \tag{34}$$

We see that we have found the missing elements in the pattern $\mathcal{S}_0$, along with two spurious elements. As a rule of thumb, tagging both (i, j) and (j, i) will increase the number of possibilities in $\mathcal{P}$ by a factor of two.

As long as the number of spurious elements in $(\mathcal{P}(\mathcal{F}))^T \cap \mathcal{P}(\mathcal{F})$ is small, the logical next step is to add the set $(\mathcal{P}(\mathcal{F}))^T \cap \mathcal{P}(\mathcal{F})$ to the original sparsity pattern and try again. In the above example, this augmented pattern will contain the true pattern and two spurious elements $(4, 5)$ and $(5, 4)$. The extra elements will both be zero in the new H computed by the CPR method; hence, they can be easily stripped from the final pattern. We thus obtain the following algorithm.

Algorithm 1. Determination of sparsity structure when a good initial guess is available

1. **Let** a symmetric trial pattern $\mathcal{S}_0$ be given.
2. **If** $\mathcal{I} \not\subseteq \mathcal{S}_0$, then set $\mathcal{S}_0 = \mathcal{S}_0 \cup \mathcal{I}$.
3. **Partition** the columns of $\mathcal{S}_0$ into groups $\{\mathcal{C}_{0,j}\}$ using a graph coloring algorithm.
4. **Perform** the CPR algorithm using $\mathcal{S}_0$ and $\{\mathcal{C}_{0,j}\}$ to produce H.
5. **Compute** the set of flaws $\mathcal{F}$ using Procedure 1.
6. **Compute** the set of possibilities $\mathcal{P}$.
7. **Define** a new pattern $\mathcal{S} = \mathcal{S}_0 + \mathcal{P} \cap \mathcal{P}^T$.
8. **Partition** the columns of $\mathcal{S}$ into groups $\{\mathcal{C}_{1,j}\}$ using a graph coloring algorithm.
9. **Perform** the CPR algorithm using the pattern $\mathcal{S}$ and partition $\{\mathcal{C}_{1,j}\}$ to produce H.
10. **Eliminate** spurious zeros from H and the final pattern $\mathcal{S}$, and exit.

Elimination of spurious zeros can be accomplished as follows.
Procedure 2: Elimination of spurious zeros

1. **Given** the matrix H and a symmetric trial pattern $\mathcal{S}$, and
2. **Given** the tolerance $\epsilon_0 > 0$,
3. **Set** $\mathcal{S}_z = \emptyset$.
4. **For all** $(i,j) \in \mathcal{S}$ with $j \geq i$ **Do**:
   - **If** both $h_{i,j}$ and $h_{j,i}$ are effectively zero:

   $$\max(|h_{i,j}|, |h_{j,i}|) < \epsilon_0, \tag{35}$$

   then add (i,j) and (j,i) to the set $\mathcal{S}_z$.
5. **Delete** all elements in $\mathcal{S}_z$ from $\mathcal{S}$ and H.
6. **Exit** procedure.

As before, the absolute test (35) can be replaced by a relative test if desired:

$$\max(|h_{i,j}|, |h_{j,i}|) < \epsilon_0 \max_{i,j} |h_{i,j}| \tag{36}$$

Limitations of Algorithm 1. Algorithm 1 works quite well as long as $nnz(\mathcal{S}_* - \mathcal{S}_0) << n$. Recall, however, that nothing precludes elements of $\mathcal{P}$ and $\mathcal{P}^T$ from matching *by accident.* If $nnz(\mathcal{S}_* - \mathcal{S}_0)$ is sufficiently small, then the number of these spurious possibilities is manageable (the user should have no qualms about doubling the size of $\mathcal{S}_0$ when computing the intermediate pattern $\mathcal{S}$). As $nnz(\mathcal{S}_* - \mathcal{S}_0) \to n$, however, the number of spurious possibilities can explode.

Suppose $nnz(\mathcal{S}_* - \mathcal{S}_0) \approx n$ for a problem with $n = 10,000$ and $nnz(\mathcal{S}_*) \approx 100,000$. Further suppose the initial pattern is colored into, say, $k = 10$ groups. On the average, we can expect two H flaws per row if we use Procedure 1. For each flaw, $\mathcal{P}$ contains about $n/k = 1000$ entries; that is, $\mathcal{P}$ is approximately 0.2 full. Now, $\mathcal{P} \cap \mathcal{P}^T$ should on the average be $(0.2)^2 = 0.04$ full. This corresponds to 4,000,000 possible entries out of the 100,000,000 entries in the full matrix. The 10,000 entries missing from the initial pattern are indeed a subset of $\mathcal{P} \cap \mathcal{P}^T$, but adding 4,000,000 elements to the pattern (400 per row) is undesirable. The situation is even worse if $k = 5$, in which case we would expect about 64,000,000 elements (6400 per row) in $\mathcal{P} \cap \mathcal{P}^T$ — practically the full matrix.

This is admittedly a back-of-the-envelope argument, but the results are supported by actual calculations. Algorithm 1 works well for $nnz(\mathcal{S}_* - \mathcal{S}_0) << n$, when the number of accidental mismatches is expected to be small, but breaks down dramatically as $nnz(\mathcal{S}_* - \mathcal{S}_0) \to n$.

4 The Multilevel Algorithm

Suppose that instead of augmenting $\mathcal{S}_0$ by $\mathcal{P}_0^T \cap \mathcal{P}_0$ after our first round of calculations, we compute $\mathcal{S}_1 = \mathcal{S}_0 + \mathcal{R}_1$ for some symmetric pattern $\mathcal{R}_1$. The purpose of $\mathcal{R}_1$ is not to make $\mathcal{S}_1$ a better pattern than $\mathcal{S}_0$, but simply to make it *different*. We then partition the new pattern, generate H_1, compute the flaws, and compute the pattern of possibilities $\mathcal{P}_1$.

Now, $\mathcal{P}_1^T \cap \mathcal{P}_1$ is unlikely to have significantly fewer spurious elements than $\mathcal{P}_0^T \cap \mathcal{P}_0$. However, the *locations* of the spurious elements will be significantly different if $\mathcal{R}_1$ has had a sufficiently randomizing effect, and the number of spurious elements in $\bar{\mathcal{P}} = (\mathcal{P}_0^T \cap \mathcal{P}_0) \cap (\mathcal{P}_1^T \cap \mathcal{P}_1)$ will be greatly reduced from the number present in $(\mathcal{P}_0^T \cap \mathcal{P}_0)$.

Continuing this process until we have reduced the number of possibilities to a manageable level gives us a *multilevel* algorithm that performs extremely well even given poor or nonexistent initial guesses for the true pattern. An initial objection one might have to this approach is the large size of the intermediate patterns in the calculation at level $\bar{l}$:

$$\bar{\mathcal{P}}_{\bar{l}} = \mathcal{P}_0 \cap \mathcal{P}_0^T \cap \mathcal{P}_1 \cap \mathcal{P}_1^T \cap \ldots \cap \mathcal{P}_{\bar{l}} \cap \mathcal{P}_{\bar{l}}^T. \tag{37}$$

Fortunately, it is not necessary to calculate or store the intermediate patterns $\mathcal{P}_i, i < \bar{l}$, nor is it necessary to store the pattern $\mathcal{P}_{\bar{l}}$. Instead, $\bar{\mathcal{P}}_{\bar{l}}$ can be computed directly from the flaws. Suppose we have an element $(i, j) \in \mathcal{P}_{\bar{l}}$, and we wish to check whether it is a member of $\mathcal{P}_l$. We first check $\mathcal{F}_l^h$ to see whether it contains an element (i, m), where column m is in the same group as column j. We then check $\mathcal{F}_l^y$ to see whether it contains an element (i, m), where column m is the group number of column j. If either of these conditions is true, then (i, j) is a member of $\mathcal{P}_l$, otherwise (i, j) is not a member. Testing (i, j) for membership in $\mathcal{P}_l^T$ is done by testing (j, i) for membership in $\mathcal{P}_l$.

Of course, appropriate arrays of pointers to the flaws must be maintained so that this test can be performed efficiently. A full discussion of possible ways to implement these procedures is beyond the scope of this paper, but we remark that one fast and convenient way to represent the flaws is to using the following arrays:

1. A sorted list of the h flaws designated by (i, j) pairs, where i and j designate the row and column indices of a flaw.

2. A sorted list of the y flaws designated by (i, k) pairs, where i is the row and k is the color of the flaw.

3. Separate lists of pointers to the locations in the above two lists corresponding to the first flaw location in any given row.

Fast sorters for these lists are also essential. In our trial implementation we used a modified heapsort.

Computation of $\bar{\mathcal{P}}_{\bar{l}}$ is formalized in the following two procedures.

Procedure 3. Test (i,j) for membership in $\mathcal{P}_l$

```
• Check to see whether $\mathcal{F}_l^h$ contains an element $(i,p)$, where column $p$ is in the same group as
  column $j$. If such an element exists, $(i,j)$ is a member of $\mathcal{P}_l$.
• Check to see whether $\mathcal{F}_l^y$) contains an element $(i,p)$, where $p$ is the group number of column
  $j$. If such an element exists, $(i,j)$ is a member of $\mathcal{P}_l$.
• If neither condition holds, $(i,j)$ is *not* a member of $\mathcal{P}_l$.
• **Exit** procedure.
```

Procedure 4. Computation of the set of possible sources for flaws, $\bar{\mathcal{P}}$

```
1. **Given** integers $\bar{l} \geq 0, m_{max} \geq 2$, and for $l = 0, 1, ..., \bar{l}$, let the partition sets $\{\mathcal{C}_{l,j}\}$ and the
   sets of flaws $\mathcal{F}_l^h$ and $\mathcal{F}_l^y$ be given.
2. **Set** $\bar{\mathcal{P}}_{\bar{l}} = \emptyset$.
3. **For** each flaw $(i,j)$ in $\mathcal{F}_{\bar{l}}^h$ **Do:**
   (a) **Set** $m = 0$
   (b) **For** each element $(i,k)$ with $k \in \mathcal{C}_{j,\bar{l}}$ and $k > i$ **Do:**
       i. **If** $(i,k)$ is not a member of $\mathcal{P}_{\bar{l}}^T$, **then go to** 4.
       ii. **For** $l = 0, \bar{l} - 1$, **Do:**
           • **If** $(i,k)$ is not a member of $\mathcal{P}_l$, **then go to** 4.
           • **If** $(i,k)$ is not a member of $\mathcal{P}_l^T$, **then go to** 4.
       iii. **The** possible source $(i,k)$ is consistent with known flaws on all levels. **Set** $m = m+1$.
       iv. **If** $m > m_{max}$, **then** too many possibilities; exit procedure immediately.
       v. **Else** add $(i,k)$ and $(k,i)$ to the set $\bar{\mathcal{P}}_{\bar{l}}$.
4. **For** each flaw $(i,j)$ in $\mathcal{F}_{\bar{l}}^y$ **Do:**
   (a) **Set** $m = 0$.
   (b) **For** each element $(i,k)$ with $k \in \mathcal{C}_{j,\bar{l}}$ and $k > i$ **Do:**
       i. **If** $(i,k)$ is not a member of $\mathcal{P}_{\bar{l}}^T$, **then go to** 5.
       ii. **For** $l = 0, \bar{l} - 1$, **Do:**
           • **If** $(i,k)$ is not a member of $\mathcal{P}_l$, **then go to** 5.
           • **If** $(i,k)$ is not a member of $\mathcal{P}_l^T$, **then go to** 5.
       iii. **The** possible source $(i,k)$ is consistent with known flaws on all levels. **Set** $m = m+1$.
```

```
        iv. If $m > m_{max}$, then too many possibilities; exit procedure immediately.
        v. Else add $(i,k)$ and $(k,i)$ to the set $\bar{\mathcal{P}}_{\bar{l}}$.
5. $\bar{\mathcal{P}}_{\bar{l}}$ has been successfully computed; exit procedure.
```

The constant m_{max} in the above procedure deserves some explanation. We have already noted that for $\bar{l} = 0$, then $\bar{\mathcal{P}}_{\bar{l}}$ can be unacceptably large. In fact, we only wish to compute $\bar{\mathcal{P}}_{\bar{l}}$ in its entirety when we have established sufficiently many levels that $nnz(\bar{\mathcal{P}}_{\bar{l}})$ is not too much greater than $nnz(\mathcal{S}_* - \mathcal{S}_0)$. Procedure 4 is halted and declared unsuccessful if, for any given flaw, more than m_{max} possible source locations exist. This causes Procedure 4 to quickly terminate without wasting computational resources if the number of possibilities is clearly too large. A suggested value for m_{max} is 5.

We can now present our *multilevel* algorithm. Since it uses Procedure 1 to determine the flaws, it *does* require an analytic expression for the diagonal of H_*. (This restrictive condition will be eliminated in the next section.)

Algorithm 2. Determination of sparsity structure given an analytic expression for the diagonal

```
1. Let a symmetric trial pattern $\mathcal{S}_0$ be given.
2. If $\mathcal{I} \not\subseteq \mathcal{S}_0$, then set $\mathcal{S}_0 = \mathcal{S}_0 \cup \mathcal{I}$.
3. Set $k = 0$.
4. Partition the columns of $\mathcal{S}_0$ into groups $\{\mathcal{C}_{0,j}\}$ using a graph coloring algorithm, and perform
   the CPR algorithm using $\mathcal{S}_0$ and $\{\mathcal{C}_{0,j}\}$ to produce $H$.
5. Compute the set of flaws $\mathcal{F}$ using Procedure 1.
6. Estimate the number of missing elements by $n_{est} = nnz(\mathcal{F}_k^y) + 0.5 * nnz(\mathcal{F}_k^h.)$
7. Attempt to compute the set of possibilities $\bar{\mathcal{P}} = \mathcal{P}_0 \cap \mathcal{P}_0^T$ using Procedure 4.
8. While $nnz(\bar{\mathcal{P}})$ is "too large" Do:
    (a) Set $k = k + 1$
    (b) Augment the original pattern with a symmetric perturbation $\mathcal{R}_k$: $\mathcal{S}_k = \mathcal{S}_{k-1} \cup \mathcal{R}_k$.
    (c) Partition the columns of $\mathcal{S}_k$ into groups $\{\mathcal{C}_{k,j}\}$ using a graph coloring algorithm.
    (d) Perform the CPR algorithm using $\mathcal{S}_k$ and $\{\mathcal{C}_{k,j}\}$ to produce $H_k$.
    (e) Compute the set of flaws $\mathcal{F}_k$ using Procedure 1.
    (f) Estimate the number of missing elements by $n_{est} = nnz(\mathcal{F}_k^y) + 0.5 * nnz(\mathcal{F}_k^h.)$
    (g) Attempt to compute the set of possibilities $\bar{\mathcal{P}} = \mathcal{P}_0 \cap \mathcal{P}_0^T \cap \mathcal{P}_1 \cap \mathcal{P}_1^T \cap \ldots \cap \mathcal{P}_k \cap \mathcal{P}_k^T$ using
        Procedure 2.
    (h) If $nnz(\bar{\mathcal{P}})$ is "sufficiently small", then exit loop.
9. Eliminate spurious zeros from $H_k$ and the pattern $\mathcal{S}_k$ using Procedure 2
10. Generate an augmented pattern $\mathcal{S}_{k+1} = \mathcal{S}_k + \bar{\mathcal{P}}$.
```

11. **Partition** the columns of $\mathcal{S}_{k+1}$ into groups using a graph coloring algorithm, and perform the CPR algorithm to produce H_{k+1}.

12. **Eliminate** spurious zeros from H and the final pattern $\mathcal{S}_{k+1}$ using Procedure 2, and exit.

To generate a symmetric perturbation pattern $\mathcal{R}_k$ with p entries, we use a random number generator to produce pairs of integers $(i,j)_k$, $k = 1, 2, ..., p/2$, with each integer being between 1 and n, and set $\mathcal{R}_k$ to be the union of these index pairs, along with the union of $(j,i)_k$, $k = 1, 2, ..., p/2$.

The size of $\mathcal{S}_k$ should be at least a small multiple of the estimated number of undetermined nonzeros, n_{est}, so we enforce

$$nnz(\mathcal{R}_k) + nnz(\mathcal{S}_{k-1}) \geq \kappa_1 n_{est}, \tag{38}$$

where $\kappa_1 \geq 2$. Also, $\mathcal{R}_k$ must be sufficiently large to effectively randomize the pattern of spurious possibilities, so we also enforce

$$nnz(\mathcal{R}_k) \geq \kappa_2 n \ , \tag{39}$$

where $\kappa_2 \geq 1$. We suggest $\kappa_1 = 2$ and $\kappa_2 = 1$, but some tradeoffs should be considered before selecting these parameters. For instance, when a poor initial guess is provided, a large value of κ_1 can lead to prohibitive storage requirements for large problems, so $\kappa_1 = 2$ may be best in this instance. If sufficient storage is available, $\kappa_1 = 4$ or higher may be more appropriate. Also, selecting larger values of κ_2 will occasionally decrease the number of levels required, but more important will increase the number of groups needed to color each level. Hence, if the cost of g evaluations is the dominant expense, small values such as $\kappa_2 = 1$ are desirable. On the other hand, if the number of groups needed to color the pattern at the highest level is increased, the computational expense of Procedure 4 will be correspondingly decreased. Hence, if g evaluations are very inexpensive, larger values such as $\kappa_2 = 5$ should be considered.

The choice of these parameters is not critical, but proper selection of the values can lead to improved efficiency.

5 The Multilevel Algorithm with Voting between Levels.

The preceding technique alleviates the main deficiency of Algorithm 1, but still has one undesirable feature: Procedure 1 requires an analytic expression for the diagonal of H_*. One way around this would be simply to flag each element of the diagonal as a flaw location at every level — clearly an inefficient approach. Fortunately, the multilevel scheme we are using provides us a way to overcome this difficulty. We can compare individual elements of H from level to level and use a voting scheme to decide which diagonal entries are probably flawed.

Suppose we have computed H_l for levels $l = 0, 1, ..., \bar{l}$, where $\bar{l} \geq 2$, and further suppose that a given element $h_{i,i}$ of the diagonal at level l has a 1 in κ chance of being flawed. If the distribution of flaws is random from level to level, then the odds of half or more of the $h_{i,i}$ elements being flawed over the levels $l = 0, 1, ..., \bar{l}$ is only about $\begin{pmatrix} \bar{l} \\ \bar{l}/2 \end{pmatrix} \kappa^{-\bar{l}}$. For $\kappa = 10$, for instance, this probability is .0006 for 4 levels, .00002 for 6 levels, etc. The probability that more than half of the elements are flawed by exactly the same amount over the various levels is even less. Hence, if the majority of elements $(h_{i,i})_l, l = 0, 1, ..., k$ take the same value, we can assume with reasonable but not perfect confidence that this value is the correct value for $h^*_{i,i}$. Note that this argument does not rely upon any assumption of an accurate initial guess for the pattern, but only upon having a sufficiently large trial pattern at each level and having each pattern be a sufficiently large perturbation of the pattern at the previous level.

For the purposes of voting on the diagonal, two elements $(h_{i,i})_l$ and $(h_{i,i})_p$ are considered equal if

$$|(h_{i,i})_l - (h_{i,i})_p| < \epsilon_v(|(h_{i,i})_l| + |(h_{i,i})_p|) \tag{40}$$

for some positive tolerance ϵ_v.

Procedure 5a: Computation of flaw locations using voting between levels (low storage version)

1. **Given** a sparse matrices $H_{\bar{l}}$ and the corresponding partition computed by the CPR algorithm using symmetric trial pattern $\mathcal{S}_{\bar{l}}$, and

2. **Given** a tolerance $\epsilon_h > 0$,

3. **Set** $\mathcal{F}_{\bar{l}}^h = \mathcal{F}_{\bar{l}}^y = \emptyset$.

4. **For** $i = 1, ..., n$ **Do**:

   - **If** a simple majority of the elements $h_{i,i}$ over the set of matrices $H_l, l = 0, 1, ..., \bar{l}$ take the same value β (using criterion (40) to perform the vote), then for $l = 0, 1, ..., \bar{l}$ **Do**
     - **If** $h_{i,i}$ is not equal to β using criterion (27) at level l, then add (i,i) to the set $\mathcal{F}_{\bar{l}}^h$.
   - **Otherwise** for $l = 0, 1, ..., \bar{l}$ **Do**
     - **Add** (i,i) to the set $\mathcal{F}_{\bar{l}}^h$.

5. **For all** $(i,j) \in \mathcal{S}_{\bar{l}}$ with $i < j$ **Do**:

   - **If** $i \neq j$ and at level $\bar{l}$ Equation (28) holds, **then** add (i,j) and (j,i) to the set $\mathcal{F}_{\bar{l}}^h$.
   - **For all** $y_{i,j}$ at level $\bar{l}$ **Do**:
     - **If** $\mathcal{S}_{\bar{l}}$ contains no element (i,k) with $k \in \mathcal{C}_j$ and Equation (29) holds, **then** add (i,j) to the set $\mathcal{F}_{\bar{l}}^y$.

6. **Estimate** the number of missing elements by $n_{est} = nnz(\mathcal{F}_k^y) + 0.5 * nnz(\mathcal{F}_k^h)$.

7. **Exit** procedure.

Although Procedure 5a is the simplest technique for computing the set of H flaws to use in our algorithm, it is not necessarily the *best* technique. A voting process can also be used with off-diagonal elements wherever such elements appear in more than one trial pattern. Although not strictly necessary, this can eliminate many of the extraneous flaws flagged by Procedure 5a, which in turn can decrease the computational expense of Procedure 4 by a substantial amount. Unfortunately, this approach requires substantially more storage.

Notice that our information obtained from voting becomes better as more levels are computed. Since computation of the flaw locations is a computationally insignificant part of our algorithm, at each level we recompute the flaws at all previous levels using the best information currently available. The Y flaws do not depend on the voting scheme, and thus do not need to be recomputed.

Procedure 5b: Computation of flaw locations using voting between levels (high storage version)

1. **Given** a set of sparse matrices H_l, $l = 0, 1, ..., \bar{l}$ and corresponding partitions computed by the CPR algorithm using symmetric trial patterns $\mathcal{S}_l$, and

2. **Given** a tolerance $\epsilon_h > 0$,

3. **Set** $\mathcal{F}_l^h = \mathcal{F}_l^y = \emptyset$.

4. **For** $i = 1, ..., n$ **Do**:

   - **If** a simple majority of the elements $h_{i,i}$ over the set of matrices $H_l, l = 0, 1, ..., \bar{l}$ take the same value β, then for $l = 0, 1, ..., \bar{l}$ **Do**
     - **If** $h_{i,i} \neq \beta$ at level l then add (i, i) to the set $\mathcal{F}_l^h$.
   - **Otherwise** for $l = 0, 1, ..., \bar{l}$ **Do**
     - **Add** (i, i) to the set $\mathcal{F}_l^h$.

5. **For** $l = 0, 1, ..., \bar{l}$ **Do:**

   - **For all** $(i, j) \in \mathcal{S}_l$ with $i < j$ **Do**:
     - **If** $i \neq j$ and at level l $|h_{i,j} - h_{j,i}| > \epsilon_h \; max(|h_{i,j}|, |h_{j,i}|)$, then
       - **If** values for $h_{i,j}$ and $h_{j,i}$ have been computed at other levels, then vote among these values to determine whether to add the index (i, j) or the index (j, i) to the set $\mathcal{F}_l^h$. If no vote is possible, or if the vote does not yield a simple majority, or if neither $h_{i,j}$ nor $h_{j,i}$ agree with the value obtained by the vote, add both (i, j) and (j, i) to the set $\mathcal{F}_l^h$.

6. **For all** $y_{i,j}$ at level $\bar{l}$ **Do**:

   - **If** $\mathcal{S}_{\bar{l}}$ contains no element (i, k) with $k \in \mathcal{C}_j$ and Condition (29) holds, **then** add (i, j) to the set $\mathcal{F}_{\bar{l}}^y$.

7. **Estimate** the number of missing elements by $n_{est} = nnz(\mathcal{F}_k^y) + nnz(\mathcal{F}_k^h.)$

8. **Exit** procedure.

Note that the Y flaws do not depend on the voting scheme and thus do not need to be recomputed.

We are now in a position to state the complete algorithm for the numerical determination of symmetric sparsity structures.

Algorithm 3. Numerical Sparsity Structure Determination

1. **Let** a symmetric trial pattern $\mathcal{S}_0$ and an estimate n_{est} of the number of missing elements be given.

2. **If** $\mathcal{I} \not\subseteq \mathcal{S}_0$, **then** set $\mathcal{S}_0 = \mathcal{S}_0 \cup \mathcal{I}$. Check $\mathcal{S}_0$ for symmetry.

3. **If** $nnz(\mathcal{S}_0) \leq \kappa_1 \; n_{est}$, then reset the initial trial pattern to be $\mathcal{S}_0 = \mathcal{S}_0 + \mathcal{R}_0$ where $\mathcal{R}_0$ is a symmetric pattern containing $\kappa_1 \; n_{est} - nnz(\mathcal{S}_0)$ elements.

4. **Set** $k = 0$.

5. **Partition** the columns of $\mathcal{S}_0$ into groups $\{\mathcal{C}_{0,j}\}$ using a graph coloring algorithm, and perform the CPR algorithm using $\mathcal{S}_0$ and $\{\mathcal{C}_{0,j}\}$ to produce H_0.

6. **While** $nnz(\bar{\mathcal{P}})$ is "too large" **Do:**

(a) **Set** $k = k + 1$.

(b) **Augment** the original pattern with a symmetric pattern $\mathcal{R}_k$ satisfying Equation (39) so that $\mathcal{S}_k = \mathcal{S}_{k-1} \cup \mathcal{R}_k$.

(c) **Partition** the columns of $\mathcal{S}_k$ into groups $\{\mathcal{C}_{k,j}\}$ using a graph coloring algorithm.

(d) **Perform** the CPR algorithm using $\mathcal{S}_k$ and $\{\mathcal{C}_{k,j}\}$ to produce H_k.

(e) **Compute** the set of flaws $\mathcal{F}_k$ using Procedure 5a or 5b.

(f) **Estimate** the number of missing elements by $n_{est} = nnz(\mathcal{F}_k^y) + nnz(\mathcal{F}_k^h.)$

(g) **If** $k < 2$, **then go to** 6(a).

(h) **Attempt** to compute the set of possibilities $\bar{\mathcal{P}} = \mathcal{P}_0 \cap \mathcal{P}_0^T \cap \mathcal{P}_1 \cap \mathcal{P}_1^T \cap ... \cap \mathcal{P}_k \cap \mathcal{P}_k^T$ using Procedure 4.

(i) **If** $nnz(\bar{\mathcal{P}})$ is "sufficiently small," **then** exit loop.

7. **End if.**

8. **Reset** $k = 0$.

9. **Generate** an augmented pattern $\mathcal{S}_0 = \mathcal{S}_0 + \bar{\mathcal{P}}$.

10. **Partition** the columns of $\mathcal{S}_0$ into groups using a graph coloring algorithm, and perform the CPR algorithm to produce H_0.

11. **Compute** the set of flaws $\mathcal{F}_0$ using Procedure 5a or 5b.

12. **Estimate** the number of missing elements by $n_{est} = nnz(\mathcal{F}_0^y) + nnz(\mathcal{F}_0^h.)$

13. **Eliminate** spurious zeros from H_0 using Procedure 2 and the pattern $\mathcal{S}_0$.

14. **If** $n_{est} = 0$, **then** exit. Otherwise, **go to** 6(a).

Some remarks concerning Algorithm 3.

1. The initial estimate of the number of missing elements n_{est} required of the user is not critical and is used only to determine an appropriate size for the default trial pattern. If the user has no idea of the true number of nonzeros in the matrix, a simple technique would be to compute a randomly selected column of ∇g using Equation (4) and $d = (0\ 0\ ...1...0)^T$, counting the number of nonzeros, and multiplying by n.

2. In Step 3 of the algorithm, we enlarge the initial pattern supplied by the user so that our trial pattern satisfies Equation (38). This is typically needed only when the user supplies no initial guess for the pattern. Although in our early experiments $\mathcal{R}_0$ was selected to be a random symmetric pattern with the requisite number of entries, a more efficient approach is to specify a band matrix of an appropriate size. This ensures that the initial trial pattern can be colored efficiently into a smaller number of groups than would be typical for a large random pattern. Furthermore, since band and multiple-band structures are extremely common, such a default initial pattern is likely to include a reasonable fraction of the true structure in any given problem.

6 Practical Implementations and Further Discussion of Numerical Results

Although Algorithm 3 is complete as presented, a few further items should be pointed out or reemphasized.

We reiterate that care must be taken with implementation, since many of the operations can be computationally intense. We have already mentioned the pointer arrays we selected to allow for fast access to the information required, but other implementers may select different data structures. Software for fast sorting of integer arrays is crucial to any implementation. We used a heapsort for most sorting tasks.

We selected Fortran 77 for our trial implementation to be easily compatible with existing code [3] for the graph coloring subproblem. Unfortunately, the variable storage requirements of our algorithm made this a poor decision. Storage capacity on a given computer was definitely the limiting factor on most of our test runs. We suggest C or Fortran 90 for future implementations.

A subtle problem can potentially arise in applications where many of the nonzeros are the same or integer multiples of each other. Consider a matrix of zeros and ones, for instance. Now, any element (i, j) that is left out of the initial trial structure will cause a flaw at some location (i, k). That is, the computed value of the (i, k) element of H will be $1 + 1 \times \frac{\delta_j}{\delta_i}$ rather than 1. Now, if $\delta_i = \delta_j$ for all i and j, we see that the computed value of *any* flawed element in H will be 2. This raises the specter of encountering flaws at both the locations (i, k) and (k, i) arising from completely different sources, yet being unable to detect either flaw because the elements are perturbed by the same amounts. Although such an occurrence is extraordinarily unlikely unless most of the nonzeros have the same values, our tests have shown that such unfortunate coincidences are not uncommon in large matrices of zeros and ones, particularly with poor initial guesses for the structure.

Fortunately there is an easy fix to this potential difficulty. In any implementation, one must merely ensure that $\delta_i \neq \delta_j$ for all i,j. For instance, if an appropriate finite difference interval for each component is $\delta_i = 10^{-6}$, one might instead use $\delta_i = \tau_i 10^{-6}$, where each τ_i is a randomly selected number between 0.5 and 2. The small change in the δs will not greatly affect the finite difference calculations, but will make the possibility of equal-magnitude symmetric flaws vanishingly small.

The choice of parameters and tolerances will affect algorithm efficiency and can in extreme cases affect reliability as well. In the results presented in §1, we used the absolute error criterion with $\epsilon_0 = 10^{-6}, \epsilon_h = 10^{-4}, \epsilon_y = 10^{-6}$, and $\epsilon_v = 10^{-6}$, and used the low-storage version of Procedure 5. The parameters κ_1, κ_2, and m_{max} were set to 2, 1, and 5, respectively. These choices worked well in almost all cases. An exception was the $n = 500,000$ case in Table 2. In this case, the sum of the effects of the elements of B in many rows became so large that the algorithm had difficulty in terminating with these tight tolerances, so they were reset to $\epsilon_0 = 10^{-2}, \epsilon_h = 10^{-1}, \epsilon_y = 10^{-1}$, and $\epsilon_v = 10^{-2}$.

All of the numerical results in this paper were obtained on a network of SPARC workstations at Argonne National Laboratory. Memory allocation was typically a limiting factor in problem size. Most of the results with n less than 50,000 were obtained on Sparcstations with 32 Mbytes of memory, but all of the larger problems were run on a Sparcstation with 256 Mbytes of memory.

Further numerical tests. Although the random sparsity patterns presented in §1 make a challenging test of our algorithm, one might wonder whether "real" structures, which tend to have periodicities, are equally amenable to solution by NSSD. We have successfully tested our algorithm on a number of real-world examples from the MINPACK-2 test collection [1], such as the composite optimal design problem 1, but for brevity we present only the most interesting.

Consider the sparsity structure shown in Figure 2, representing the fairly complex sparsity pattern arising from the solution of the Ginzburg-Landau equations in two dimensions for superconducting materials [1],[7]. Unlike the random test structures presented in §1, this problem has a structure typical of many real applications. Furthermore, a relatively small portion of nonzeros falls on the central band of the matrix, so that the internal initial guess used by NSSD will not be exceptionally good. Evaluation of each gradient is relatively inexpensive, involving $o(n)$ operations.

An interesting feature of this problem is that the Hessian matrix evaluated at the standard initial guess has a large number of nonzero elements that are nevertheless very small. For instance, a typical size of a nonzero element is about 2.5 in the $n = 40,000$ case, but several thousand of the 457,094 "nonzeros" have magnitudes less than 10^{-5}. Since this is roughly the same magnitude as errors induced by the use of finite differences, an appropriate choice of tolerances for this problem will be expected to miss these small elements. Hence, care must be taken in interpreting the results.

In our test we selected the relative error criterion, chose $\epsilon_0 = 10^{-6}, \epsilon_h = 10^{-4}, \epsilon_y = 10^{-6}$, and $\epsilon_v = 10^{-6}$, and used the high-storage version of Procedure 5. The parameters κ_1, κ_2, and m_{max} were set to 2, 1, and 5, respectively. The results are shown in Table 4. In each case we correctly determined the location of all the nonzeros in the structure with the exception of elements with size less than 10^{-5}.

Table 4. Performance of NSSD, and comparison to full finite differences, for the 2-D superconductivity problem with no initial guess for the pattern provided by user.

n	nnz	Number of g evals.	g evaluation time (sec)	Total NSSD time (sec)	Estimated time for a full finite difference Hessian to get structure ($n\times$ cost(g eval)) (sec)
400	4,592	199	3.57	8.06	7.17
1,600	18,372	244	14.8	42.5	97.05
2,500	28,734	326	30.8	81.6	236.2
3,600	41,370	294	39.4	138.8	482.4
6,400	73,470	305	72.5	257.3	1,521.3
10,000	114,764	340	124.1	550.1	3,650.
14,400	165,118	298	151.6	987.7	7,325.
25,600	293,006	359	315.9	2,761.9	22,526.
40,000	457,094	344	473.1	4,845.9	55,012.

So far we have based our performance analysis on a comparison with full finite differences. Another basis for comparison is the amount of time required to solve the associated optimization problem once the structure is known. Once the sparsity pattern has been determined for this 2-D superconductivity test problem with n=40,000, we used a graph-coloring-Hessian truncated-Newton conjugate-gradient code to solve the associated optimization problem in roughly 2 to 4 times the time required to run NSSD with no a priori information. The expense of NSSD is therefore not negligible but is certainly not prohibitive either.

7 Conclusions

7.1 Summary

By examining asymmetries in the results of the nonsymmetric CPR algorithm applied to a symmetric problem, one can often identify the location of possible nonzeros that were omitted from the assumed structure if the assumed structure is close to the true structure. By adding a multilevel approach and cross-correlating results using several different trial patterns, one can quickly identify the location of all nonzeros even if no initial guess is available. The technique also effectively filters out elements of small size.

When gradients are expensive, this algorithm can be several orders of magnitude faster than using a full finite difference calculation. Even if gradients are inexpensive it typically outperforms the brute force approach by factors of 10 to 100 for sufficiently large problems.

The algorithms presented in this paper are in no sense mature, and it is quite likely that they can be simplified and refined substantially. The algorithms and numerical results in this paper are simply our initial attempt to implement the central ideas presented in §2 — §5. Other researchers are encouraged to try their own implementations. Both reliability and speed are factors in judging a given implementation.

7.2 Further Ideas

The possibilities for further work on this topic are extraordinarily rich. Without going into detail, we would mention the following possible refinements and extensions.

Avoidance of voting on diagonal elements. The least straightforward part of our algorithm is the procedure for spotting flaws on the diagonal by voting between levels. A possible technique to avoid such voting is to specifically *exclude* all diagonal elements from each trial sparsity pattern until the final set of calculations. Such an approach might at first seem counterintuitive, since almost all real problems have mostly nonzero diagonal elements a priori, but recall that there is no real requirement that our algorithm always use the best possible approximation at each iteration. The elimination of the necessity of voting on diagonal elements may offset any loss of efficiency incurred by not using the best guess for the structure each time.

Band matrix formulation. Rather than using different random structures at each level until the number of possible sources is winnowed to a reasonable number, one can instead use a sequence of band matrices of increasing bandwidth, where the bandwidth at each level is relatively prime with respect to previous levels. (For instance, the initial guess might be a band structure of width 5, the second level might use a banded trial structure with bandwidth 9, while the third level might have bandwidth 13, and so on.) Although this approach gives little homage to the idea of always using the best possible guess for the structure, it has a number of advantages. First, band structures have trivial, periodic coloring, which reduces the computational burden. Second, storage/retrieval schemes for elements of band matrices are typically much faster than for the list-format storage of general sparse matrices. Third, if most of the matrix is band limited and falls within one of the trial sparsity structures, then the remaining portion can in principle be reconstructed with much less overhead than in the algorithm presented in this paper. Suppose, for instance, that only one flaw exists in a given row at each level, and our trial bandwidths at the different levels are the primes 3,5,7,13, ... Using elementary linear diophantine equations, one can easily see that the separation between possible flaw sources in this row is exactly the product $3 \times 5 \times 7 \ldots$ up to the prime bandwidth corresponding to the highest level so far. Since this separation grows *very* fast, the computational overhead for such an approach on a limited problem class might drop almost to

$o(n \ln n)$ with an appropriate modification of Procedures 3 and 4. This approach is also much more amenable to proofs.

Parallel computation. Many of the most intensive computations in this algorithm seem in principle quite amenable to parallelization. In particular, distributed processing looks attractive, since total memory available was the limiting factor in the size of our test problems when running on sequential machines. Nevertheless, the data structures involved in a distributed implementation may be very challenging.

Nonsymmetric problems. Possibly the most exciting extension of this work involves computation of nonsymmetric sparsity patterns. At first glance, this might appear impossible, since our method depends so heavily on symmetry. However, the multilevel voting procedure introduced to help identify diagonal flaws can be extended directly to the nonsymmetric case. Specifically, one identifies flaws *only* by voting between levels and not by comparing between symmetric pairs of elements. Similarly, one would compute the sets of possibilities $\bar{\mathcal{P}} = \mathcal{P}_0 \cap \mathcal{P}_1 \cap ... \cap \mathcal{P}_k$ rather than $\bar{\mathcal{P}} = \mathcal{P}_0 \cap \mathcal{P}_0^T \cap \mathcal{P}_1 \cap \mathcal{P}_1^T \cap ... \cap \mathcal{P}_k \cap \mathcal{P}_k^T$. While not as efficient as the symmetric approach, this idea has much promise.

References

[1] B. Averick, R. Carter, and J. Moré, *The MINPACK-2 test problem collection (preliminary version)*, Tech. Report 91-76, AHPCRC, University of Minnesota, 1991. Also report ANL/MCS-TM-150.

[2] T. Coleman and J. Cai, *The cyclic coloring problem and estimation of sparse Hessian matrices*, SIAM J. Alg. Disc. Meth., 7 (1986), pp. 221–235.

[3] T. F. Coleman, B. S. Garbow, and J. J. Moré, *Software for estimating sparse Jacobian matrices*, ACM Trans. Math. Software, 10 (1984), pp. 329–345.

[4] T. Coleman and J. Moré, *Estimation of sparse Jacobian matrices and graph coloring problems*, SIAM J. Numer. Anal., 20 (1983), pp. 187–209.

[5] ——, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, 28 (1984), pp. 243–270.

[6] A. Curtis, M. Powell, and J. Reid, *On the estimation of sparse Jacobian matrices*, IMA J. Appl. Math., 13 (1974), pp. 117–120.

[7] J. Garner, M. Spanbauer, R. Benedek, K. Strandburg, S. Wright, and P. Plassmann, *Critical fields of Josephson-coupled superconducting multilayers*, Physical Review B (to appear,1992).

[8] M. Powell and P. L. Toint, *On the estimation of sparse Hessian matrices*, SIAM J. Numer. Anal., 16 (1979), pp. 1060–1074.