Calculating the Breakdown Point of Sparse Linear Models

Jung Jin Cho
Baker Hughes Inc.
Houston, TX 77019

Yong Chen
Department of Mechanical and Industrial Engineering
The University of Iowa
Iowa City, IA 52242

Yu Ding
Department of Industrial and Systems Engineering
Texas A&M University
College Station, TX 77843-3131
(*yuding@iemail.tamu.edu*)

In robust statistics, the concept of breakdown point was introduced to quantify the robustness of an estimator in a linear regression model. Computing the breakdown point is useful in tuning some robust regression estimators (e.g., the least trimmed squares estimator). Computing the breakdown point for a structured linear model (i.e., one with dependencies among some p rows of the $n \times p$ design matrix $\mathbf{X}$) can be very demanding. This article presents an algorithm for calculating the maximum breakdown point for sparse linear models, which are a special type of structured linear model whose design matrix has many zero entries. The algorithm decomposes a sparse design matrix into smaller submatrixes on which the computation is performed, thereby leading to substantial savings in computation. An assembly process, along with a few numerical examples, illustrate the application of the algorithm and demonstrate its computational benefits.

KEY WORDS: Breakdown point; Least trimmed squares estimator; Robust estimation; Structured linear regression.

1. INTRODUCTION

The research effort reported in this article is concerned with robust estimators for a linear regression model. Consider a general linear regression model,

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e}, \tag{1}$$

where $\mathbf{y} = (y_1, \dots, y_n)^T$ is an n-dimensional vector of measurements or observations, $\boldsymbol{\beta} = (\beta_1, \dots, \beta_p)^T$ is a p-dimensional vector of unknown parameters, $\mathbf{e} = (e_1, \dots, e_n)^T$ is the random errors, and $\mathbf{X} = (\mathbf{x}_1^T, \dots, \mathbf{x}_n^T)^T$ is the $n \times p$ design matrix. Typically, the error term $\mathbf{e}$ is assumed to be normally distributed with mean $\mathbf{0}$ and covariance matrix $\sigma^2\mathbf{I}$. $\boldsymbol{\beta}$ is commonly estimated using the least squares (LS) estimator, $\hat{\boldsymbol{\beta}}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$. One major drawback of the LS estimator is its lack of robustness; heavy-tailed distributions in $\mathbf{e}$ and outliers in $\mathbf{y}$ or modeling errors in $\mathbf{X}$ prohibit accurate estimation of the coefficients in $\boldsymbol{\beta}$ using an LS estimator (Wilcox 2005).

The outlier sensitivity of LS estimation has stimulated interest in developing robust regression estimators. To quantify the robustness of a regression method, Huber (1983) introduced the concept of the finite-sample breakdown point, which we will briefly review in the sequel. Let $\mathcal{Z} = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ denote the collection of known or observed information contained in $\mathbf{y}$ and $\mathbf{X}$ and $T(\mathcal{Z})$ denote a regression estimator using $\mathcal{Z}$; thus $\hat{\boldsymbol{\beta}} = T(\mathcal{Z})$. When data points in $\mathcal{Z}$ are contaminated by outliers, the estimation of $\boldsymbol{\beta}$ may deviate substantially from its supposed true value. Suppose that m data points of $\mathcal{Z}$ are contaminated, meaning that any m data points can be replaced by arbitrary values. We designate the contaminated data set $\mathcal{Z}'_m$. The resulting estimator is $T(\mathcal{Z}'_m)$. The maximum difference between $T(\mathcal{Z})$ and $T(\mathcal{Z}'_m)$ is denoted by bias$(m; T)$, defined as

$$\operatorname{bias}(m; T) = \sup_{\mathcal{Z}'_m} \|T(\mathcal{Z}'_m) - T(\mathcal{Z})\|, \tag{2}$$

where the supremum is over all possible $\mathcal{Z}'_m$ for a given m, and $\|\cdot\|$ represents the Euclidean norm. The breakdown point of the regression estimator T, denoted by $\epsilon_n^*(T, \mathcal{Z})$, is defined as

$$\epsilon_n^*(T, \mathcal{Z}) = \min\left\{\frac{m}{n} : \operatorname{bias}(m; T) \text{ is infinite}\right\}. \tag{3}$$

Intuitively, the foregoing breakdown point is the minimal fraction of gross outliers in the data that could cause the estimator to give totally wrong results. The higher the breakdown point, the more outliers an estimator can tolerate, and thus the more robust it is.

The breakdown point associated with a robust regression depends not only on the estimator T, but also on the linear dependence relationships among the rows in the design matrix $\mathbf{X}$ (Coakley and Mili 1993; Davies 1993; Mili and Coakley 1996). Throughout the article, we call a design matrix $\mathbf{X}$ *structured* if some p row vectors in $\mathbf{X}$ are linearly dependent and *unstructured* if any p row vectors of $\mathbf{X}$ are linearly independent. Consequently, model (1) is a *structured* linear regression when $\mathbf{X}$ is structured.

Our previous experience provides the following example of a structured linear model. Figure 1 shows a multistation assembly

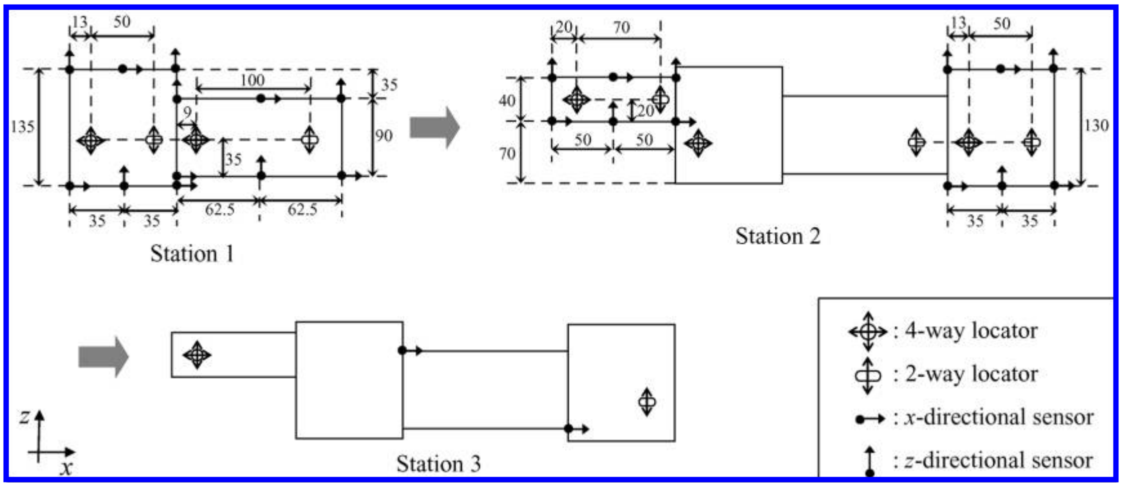


Figure 1. A three station assembly process.

process, equipped with a distributed, redundant sensor system. This assembly process consists of three stations. At Station 1, two parts are assembled, and the resulting subassembly is transferred to Station 2. At Station 2, the subassembly is assembled with two more parts. At Station 3, the final assembly is positioned for quality inspection. Coordinate sensors are placed not only at Station 3, but also at Stations 1 and 2.

A coordinate sensor measures a dimensional deviation of the part in either the x-direction or the z-direction. The response $\mathbf{y}$ includes the measurements obtained by the $n = 26$ sensors. The positions and measurement directions of the sensors are indicated by arrows in Figure 1. The unknown parameter $\boldsymbol{\beta}$ denotes the deviations associated with the fixture locators that hold the parts during the assembly operation. Each part or each finished subassembly is positioned by a pair of locators, consisting of a four-way locator that controls the part motions in both x- and z-directions and a two-way locator that controls the part motion in only the z-direction. The potential locator deviations to be estimated are represented by a double-arrowed bar on a circle or on a slot that indicates the location of a fixture locator. There are a total of $p = 12$ potential deviations at the three stations to be estimated, and the dimensions of $\boldsymbol{\beta}$ are 12×1.

In the engineering literature (e.g., Jin and Shi 1999) a linear model, in the same form as Equation (1), is commonly used to establish a connection between the sensor output $\mathbf{y}$ and the locator deviation $\boldsymbol{\beta}$ for such an assembly process. The design matrix $\mathbf{X}$ of this system is structured, because many sets of p rows in $\mathbf{X}$ are linearly dependent. (We show this $\mathbf{X}$ in Section 4 when we revisit this example.) The design matrix is not only structured, but also *sparse*, meaning that it has many zero entries; in fact, about 75% of the entries are 0. When a robust estimator is used in such an application, how the linear dependencies among the rows in the design matrix will affect the breakdown point and the choice of parameters in a robust regression procedure need to be considered.

Mili and Coakley (1996) provided a rather comprehensive account of robust estimation for a structured linear regression. They stated that for a structured linear model, the upper bound for breakdown point, or the maximum breakdown point, is $\epsilon^*_{\max,n}$ such that

$$\epsilon^*_{\max,n} = \frac{\lfloor (n - M + 1)/2 \rfloor}{n}, \tag{4}$$

where $\lfloor a \rfloor$ denotes the largest integer $\le a$, and M can be considered equal to $n - d^*$, where d^* is the minimum number of rows of $\mathbf{X}$ that, if deleted, makes $\mathbf{X}$ singular. Here d^* can be interpreted as the *redundancy degree of measurements*, a concept that has been used in engineering applications (Stanley and Mah 1981; Luong et al. 1994; Staroswiecki, Hoblos, and Aïtouche 2004). As such, the quantity M complements the redundancy in a design matrix.

The Least Trimmed Squares (LTS) estimator, developed by Rousseeuw (1984), is in the class of high breakdown point estimators, capable of attaining the maximum breakdown point $\epsilon^*_{\max,n}$. An LTS estimator is determined through

$$\min \sum_{i=1}^{h} w^2_{(i)}, \tag{5}$$

where h is called a trimming parameter and $w^2_{(1)} \le w^2_{(2)} \le \cdots \le w^2_{(n)}$ are the squared residuals $(y_i - \mathbf{x}_i\hat{\boldsymbol{\beta}})^2$ for $i = 1, \ldots, n$, arranged in ascending order. To guarantee that an LTS estimator achieves the maximum breakdown point $\epsilon^*_{\max,n}$, Mili and Coakley (1996) proved that h must be chosen such that

$$h_L \le h \le h_U, \tag{6}$$

where $h_L = \lfloor (n + M + 1)/2 \rfloor$ and $h_U = \lfloor (n + M + 2)/2 \rfloor$. The foregoing condition implies that in general, making h smaller may improve robustness by dropping outliers from the fit, but making h too small can in fact decrease the breakdown point. Thus the benefit of knowing M is in determining the proper choice of h.

For an unstructured design matrix, $d^* = n - p + 1$, so that $M = p - 1$, meaning that M can be determined based solely on the dimensions of a design matrix. Considering the foregoing assembly example, suppose that the design matrix is unstructured. Then its degree of redundancy is $d^* = 15$, and $M = 11$, so that the maximum breakdown point, $\epsilon^*_{\max,n} = \frac{8}{26}$, is achieved if an LTS estimator is used with $h = 19$.

In fact, the design matrix of the assembly example is structured so that its degree of redundancy, d^*, is much smaller than 15 because of cancellations resulting from between-row linear dependencies. (Our later analysis shows that $d^* = 5$ for the foregoing example.) In turn, a smaller d^* makes M larger and the maximum breakdown point smaller. To choose a proper h and assess the maximum breakdown point, computing M (or, equivalently, d^*) for a structured linear regression is critical.

The foregoing line of argument regarding the choice of h is made in an attempt to attain the maximum breakdown point, an important robustness consideration. Keep in mind that other robustness considerations also need to be taken into account when choosing h, the discussion of which is deferred to Section 3.

Calculating M for a structured linear model can be computationally expensive, however. Improving the computational efficiency does not appear to be straightforward. The algorithm for finding M was not given explicitly by Mili and Coakley (1996). Previous authors have found that computing M for a structured linear model is in fact an nondeterministic polynomial-time hard problem (Vardy 1997; Cho, Chen, and Ding 2007b).

In this work we study a special case of structured linear models, those that are (or can be converted into) sparse linear models. Sparse linear models are encountered in many applications in the engineering field and physical sciences. The foregoing example is one such application; we discuss others in Section 2.3.

For a sparse linear model, we can devise a decomposition algorithm for calculating M efficiently. This algorithm uses the linear dependencies among the rows in a sparse $\mathbf{X}$ and decomposes the original design matrix into smaller submatrixes. Performing the computation on the submatrixes should provide substantial computational savings compared with alternatives that deal with the original matrix. This algorithm is based on a decomposition property established for a connected matroid by Cho, Chen, and Ding (2007b), but we attempt to present our algorithm without involving the concepts from matroid theory. For a certain class of nonsparse structured design matrixes, we provide a transformation procedure that will convert these into sparse matrixes, allowing the use of the proposed decomposition algorithm. Of course, our method has its own limitations; some conditions on the sparsity of the design matrixes are needed to provide the promised computational savings. We discuss the conditions and their implications after presenting our method.

The rest of the article is organized as follows. Section 2 presents the decomposition algorithm for computing M and discusses its applicability conditions. Section 3 explores how to set the trimming parameter for an LTS estimator. Section 4 revisits the multistation assembly process and illustrates the application of the algorithm in finding the maximum breakdown point. Section 5 concludes the article.

2. CALCULATING M

Given the relationship between d^* and M, a rudimentary exhaustive procedure that can calculate M of a design matrix $\mathbf{X}$ is to test the rank of $\mathbf{X}$ after removing a certain number of rows. In fact, this is the procedure used in engineering applications to compute the degree of redundancy d^* (Stanley and Mah 1981; Luong et al. 1994; Staroswiecki, Hoblos, and Aïtouche 2004). The specific procedure is as follows, where $\mathbf{X}_{(-d)}$ is the reduced design matrix after d rows are removed:

1. Set $d = 1$.
2. If there exists $\mathbf{X}_{(-d)}$ such that $\operatorname{rank}(\mathbf{X}_{(-d)}) < \operatorname{rank}(\mathbf{X})$, stop. The corresponding d is d^*. Then $M = n - d^*$.
3. $d = d + 1$, and return to step 2.

Using this *exhaustive rank testing* procedure can be computationally expensive when $\mathbf{X}$ is large and M is small, however.

The basic idea of our decomposition algorithm is as follows. If a sparse $\mathbf{X}$ matrix can be decomposed into smaller submatrixes, then rank testing of the submatrixes can be done much faster than testing on the original matrix. To an extreme, if $\mathbf{X}$ comprises a set of $r > 1$ disjoint submatrixes, $\mathbf{B}_1, \ldots, \mathbf{B}_r$, as illustrated in Figure 2(a), then d^* is simply the smallest value of the d^*'s associated with $\mathbf{B}_i$ for $i = 1, \ldots, r$, and then M can be computed accordingly. This matrix format is called a *block diagonal form*. In terms of regression, when a design matrix is in a block diagonal form, r linear regression problems can simply be solved separately.

Of course, a design matrix in a sparse linear regression may not take a block diagonal form. A more common manifestation of a structured design matrix is a *bordered block diagonal form* (BBDF), as shown in Figure 2(b), where nonzero submatrixes, $\mathbf{B}_i$ for $i = 1, \ldots, r$, are called blocks and $\mathbf{S} = (\mathbf{S}_1 \ \cdots \ \mathbf{S}_r)$ is the

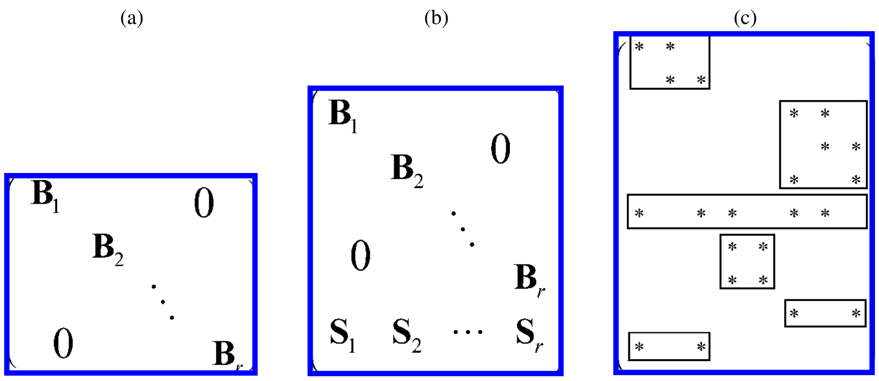


Figure 2. The design matrix structures. [In (c), "*" denote nonzero elements in the matrix. Elsewhere, the elements are 0s.] (a) Block diagonal structure; (b) bordered block diagonal structure; (c) hidden block diagonal structure.

border submatrix. The rows of **S** have nonzero elements in the columns of at least two diagonal blocks. (See Aykanat, Pinar, and Çatalyürek 2004 for more discussion on the BBDF.) In the sequel, we refer to the rows in the border submatrix as simply *border rows*. The common feature of BBDF matrixes is that once the border rows are identified and removed, the rest of the matrix can be decomposed into smaller disjoint submatrixes, on which rank testing can be performed and computational effort can be reduced.

A sparse design matrix established solely from specific applications may present a complicated appearance, such as that shown in Figure 2(c), which looks different than a BBDF. A sparse matrix **X** can be transformed into a BBDF (Aykanat, Pinar, and Çatalyürek 2004); however, such a transformation is not straightforward. In Section 2.1 we present a sophisticated procedure based on graph theory devised to fulfill this transformation. A nonsparse matrix also may be transformed into a BBDF; we defer the discussion of this to Section 2.3.

As we discuss later, M cannot always be determined by simply using the smaller submatrixes after decomposition. We demonstrate that this can be done only when M is smaller than a bound or, equivalently, when d^* is known to be greater than a bound. We show that the bound is determined by the number of border rows and the number of blocks of **X**.

In light of the foregoing, we present a *bound-and-decompose* algorithm. This bound-and-decompose algorithm applies the exhaustive search algorithm until d reaches a bound. Then, it decomposes the original matrix and also intelligently decides which combination of the resulting submatrixes should be tested to gain the greatest saving in computation. We give the details of this bound-and-decompose algorithm in Section 2.2.

In Section 2.3 we present a transformation for possibly converting a nonsparse structured matrix into a sparse one without affecting its M value. This transformation is based on the *reduced row echelon form* (Lay 1997) of a matrix. We also discuss the conditions under which the proposed decomposition method can provide the promised computational savings and explore their practical implications.

Figure 3 summarizes the aforementioned procedure for finding M. Note that the steps shown in the Figure 3 are associated with the subsections in Section 2.

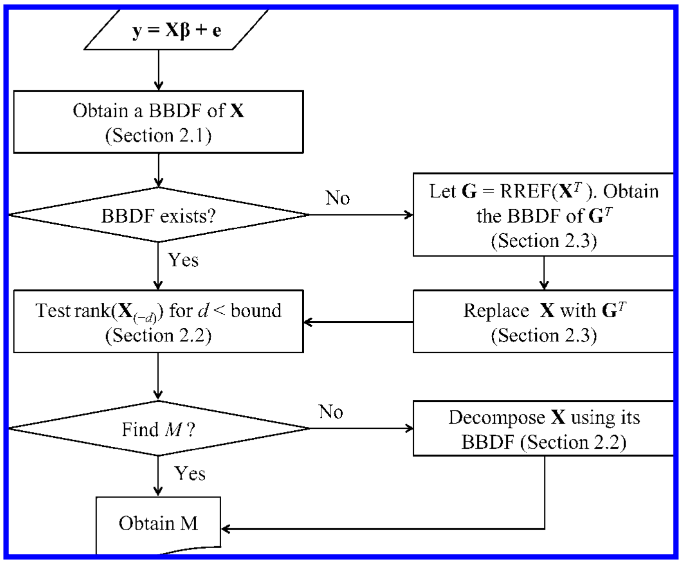

y = Xβ + e
Obtain a BBDF of X (Section 2.1)
BBDF exists?
No
Yes
Let G = RREF(X^T). Obtain the BBDF of G^T (Section 2.3)
Test rank(X_(−d)) for d < bound (Section 2.2)
Replace X with G^T (Section 2.3)
Find M?
No
Yes
Decompose X using its BBDF (Section 2.2)
Obtain M

Figure 3. Overview of the steps involved in calculating M.

2.1 Graph-Based Decomposition of **X** Matrix

Here we present a procedure that decomposes a sparse design matrix into blocks and border rows. In graph theory, a matrix is typically represented by a *bipartite graph*. Existing methods in graph theory, such as the algorithm of Even (1979), can help fulfill the foregoing decomposition objective. Consequently, we start by introducing a bipartite graph representation of the matrix and the relevant graph-related concepts and methods. Decomposition based on a bipartite graph does not guarantee the production of border rows, however; it could produce border columns, which are not useful in the current context. For this reason, we introduce a second graph, a *row-only graph* on top of the bipartite graph, to handle this technical detail.

A *bipartite graph*, also called a *bigraph*, is a graph whose vertexes can be partitioned into two disjoint sets, represented by V^+ and V^-, such that no arc exists between any two vertexes within the same set. Given **X**, let Row(**X**) and Col(**X**) denote its row set and column set, that is, $\mathbf{X} = (x_{ij}|i \in \text{Row}(\mathbf{X}), j \in \text{Col}(\mathbf{X})$, where x_{ij} is the (i,j) entry. A bipartite graph on **X** is a graph $G(V^+, V^-, A)$ with a vertex set $V^+ = \text{Row}(\mathbf{X})$ and $V^- = \text{Col}(\mathbf{X})$ and an arc set $A = \{(i,j)|x_{ij} \neq 0\}$. By definition, each arc has the initial vertex in Row(**X**) and the terminal vertex in Col(**X**). Figure 4(b) shows the bipartite graph representation of the design matrix given in Figure 4(a).

The decomposition procedure must first identify a *separating set* of the bigraph. A separating set, S, is defined as the set of vertexes if there exist two vertexes, $a, b \notin S$, such that all paths between a and b pass through at least one vertex of S (Even 1979, p. 121). Even (1979, pp. 121–129) provided a detailed algorithm for obtaining the smallest separating set. To find border rows (not border columns), we need to restrict the final separating S to be a subset of Row(**X**), but directly applying the algorithm of Even (1979) does not guarantee this.

To handle this problem, we introduce a second graph, a row-only graph, which can be derived from the bigraph as follows: Add a new arc between any two vertexes in Row(**X**) that are connected through a vertex in Col(**X**), and then delete the vertexes in Col(**X**) and their associated arcs. Figure 4(c) shows the row-only graph created from the bigraph in Figure 4(b). Then the algorithm of Even (1979) can be applied to the row-only graph to identify an S that is a subset of Row(**X**).

Once a separating set S that corresponds to the border rows is identified, the next step is to partition the rest of the matrix into disjoint submatrixes. Let $\mathbf{X}[I, J]$ denote the submatrix of **X** with the row set I and the column set J, namely, $\mathbf{X}[I, J] = (x_{ij}|i \in I, j \in J)$; also let $R = \text{Row}(\mathbf{X})$ and $C = \text{Col}(\mathbf{X})$. The notation $\mathbf{X}[R - S, C]$ represents the rest of the original **X** matrix after removal of a separating set S. To identify each disjoint submatrix of $\mathbf{X}[R - S, C]$, the depths-first search (DFS) algorithm can be applied to the disconnected bipartite graph $G(R - S, C, A_r)$, where A_r denotes the arc set $\{(i,j)|x_{ij} \neq 0, i \notin S\}$. (For the details of the DFS algorithm, please refer to Even 1979, pp. 53–57.)

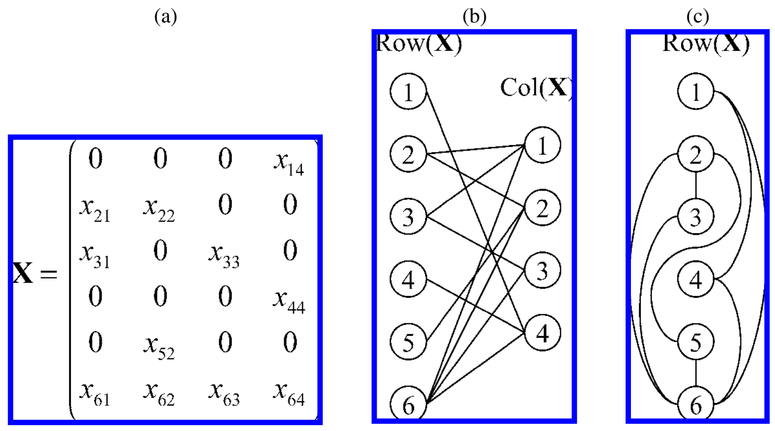

(a)
(b)
(c)
$\mathbf{X} = \begin{pmatrix} 0 & 0 & 0 & x_{14} \\ x_{21} & x_{22} & 0 & 0 \\ x_{31} & 0 & x_{33} & 0 \\ 0 & 0 & 0 & x_{44} \\ 0 & x_{52} & 0 & 0 \\ x_{61} & x_{62} & x_{63} & x_{64} \end{pmatrix}$
Row(**X**)
Col(**X**)
Row(**X**)

Figure 4. A bipartite graph representation of **X**, where x_{ij} represents a nonzero element at ith row and jth column. (a) The design matrix; (b) the associated bipartite graph; (c) the row-only graph.

Combining the foregoing two steps, the procedure for identifying the structure of a design matrix can be summarized as follows:

1. For all $v_1, v_2 \in \text{Row}(\mathbf{X})$ and $v_1 \neq v_2$, if there exists $v_3 \in \text{Col}(\mathbf{X})$ such that v_1 and v_2 are connected through v_3, then create a new arc connecting v_1 and v_2.
2. Delete all of the nodes corresponding to $\text{Col}(\mathbf{X})$ and their associated arcs. Then we have a row-only graph with the vertexes corresponding to $\text{Row}(\mathbf{X})$.
3. Using the algorithm of Even (1979), find the smallest separating set Q_1 of the row-only graph from step 2.
4. Find $v_c \in \text{Col}(\mathbf{X})$, which has the fewest number of neighbors in the bipartite graph. Let $Q_2 \subseteq \text{Row}(\mathbf{X})$ be the set of all of the neighbors of v_c.
5. If $|Q_1| \leq |Q_2|$, then $S = Q_1$; otherwise, $S = Q_2$.
6. Run DFS on $G(\text{Row}(\mathbf{X}) - S, \text{Row}(\mathbf{X}), A_r)$ to find the blocks.

In the foregoing procedure, after applying the algorithm of Even (1979) in step 3, the resulting smallest separating set of the row-only graph generally is also the smallest separating subset of $\text{Row}(\mathbf{X})$ for the original bigraph. But there can be an exception to this, as captured by steps 4 and 5. By the definition of a bigraph, Q_2, obviously a subset of $\text{Row}(\mathbf{X})$, is also a separating set. Q_2 could be smaller than Q_1. If that were the case, then step 5 could simply select the smaller of Q_1 and Q_2. It is not difficult to prove that S in step 5 is the smallest separating subset of $\text{Row}(\mathbf{X})$ for the bipartite graph.

Consider, for example, the design matrix shown in Figure 4. Figure 5 illustrates the foregoing procedure. First, the sixth row is identified as the separating set, which is also the border row. After the sixth row is removed, the DFS algorithm will decompose the rest of the bipartite graph into two disconnected subgraphs that represent two disjoint submatrixes, $\mathbf{X}[\{2,3,5\},\{1,2,3\}]$ and $\mathbf{X}[\{1,4\},\{4\}]$, which are the blocks in the BBDF.

In terms of permuting a sparse matrix into a bordered block form, several research groups in parallel computing have developed other effective methods (e.g., Ferris and Horn 1998; Aykanat, Pinar, and Çatalyürek 2004). Ferris and Horn (1998) used a two-phase approach, which first transforms a matrix into a format with both border rows and border columns and then replaces the border columns with border rows using various heuristics. Aykanat, Pinar, and Çatalyürek (2004) modeled the matrix using a hypergraph and decomposed the matrix through partitioning of the corresponding hypergraph. These approaches are quite involved, and their use requires several inputs, such as the number of blocks and balancing criteria for regulating relative sizes of blocks, which may not seem intuitive to practitioners outside the area of parallel computing. In

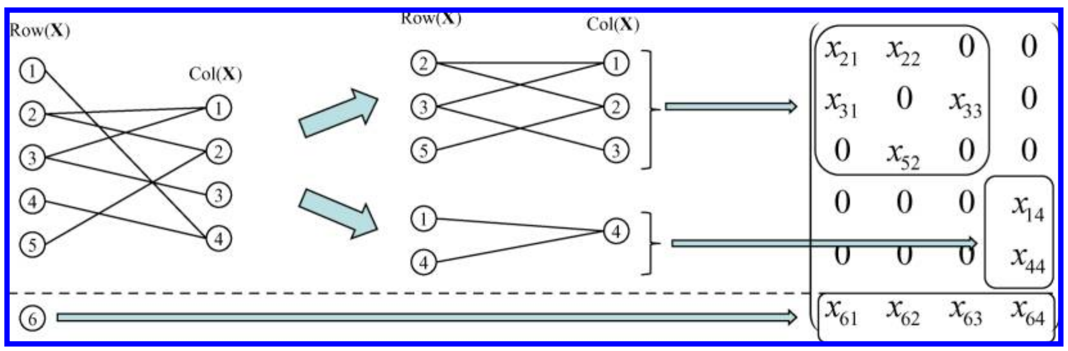

Row(**X**)
Col(**X**)
Row(**X**)
Col(**X**)
$\begin{pmatrix} x_{21} & x_{22} & 0 & 0 \\ x_{31} & 0 & x_{33} & 0 \\ 0 & x_{52} & 0 & 0 \\ 0 & 0 & 0 & x_{14} \\ 0 & 0 & 0 & x_{44} \\ x_{61} & x_{62} & x_{63} & x_{64} \end{pmatrix}$

Figure 5. In (a), node "6" has been identified as the separating set and removed from the rest of the bipartite graph. Moving from (a) to (b) shows the identification of blocks on the bipartite graph after removal of the separating set. In (b), x_{ij} corresponds to the matrix entry in Figure 4 but is placed at a permuted position in the resulting BBDF matrix.

comparison, our approach is easier to understand and simpler to implement. It automatically identifies the structure in a matrix and yields a matrix decomposition with the smallest border size, which also can be found by the hypergraph partitioning method when no balancing is enforced.

2.2 Finding *M*

Suppose that applying the foregoing decomposition procedure yielded r disconnected subgraphs (disjoint blocks). Let the set of vertexes in V^- associated with each these subgraphs be denoted by $(C_1, \dots, C_r)$, which correspond to the columns for each block, and let R_i denote the nonzero row labels of $\mathbf{X}[R, C_i]$. Then we partition the original matrix into submatrixes $\mathbf{X}[R_1, C_1], \dots, \mathbf{X}[R_r, C_r]$. Note, however, that $\mathbf{X}[R_1, C_1], \dots, \mathbf{X}[R_r, C_r]$ are not necessarily disjoint, because they contain the rows from the separating set S. For example, in Figure 4, $(C_1, C_2) = (\{4\}, \{1, 2, 3\})$, which can be identified directly from the partitioned bigraph in Figure 5. The $(R_1, R_2) = (\{1, 4, 6\}, \{2, 3, 5, 6\})$, which include the border row $\{6\}$, so that R_1 and R_2 are not disjoint.

For submatrix testing, we can test not only a submatrix that is an individual block, but also a submatrix that combines several individual blocks, that is, a combination of $\mathbf{X}[R_i, C_i]$ for $i = 1, \dots, r$. Let $\mathcal{X}^{(k)}$ denote a collection of submatrixes of $\mathbf{X}$ as

$$\mathcal{X}^{(k)} = \left\{ \mathbf{X}\left[\bigcup_{i \in U} R_i, \bigcup_{j \in U} C_j \right] : \forall U \subseteq \{1, 2, \dots, r\} \text{ s.t. } |U| = k \right\}.$$

For a given k, there are C_r^k elements in $\mathcal{X}^{(k)}$; a submatrix $\mathbf{X}^{(k)} \in \mathcal{X}^{(k)}$ is called a k-block submatrix. Obviously, the collection of one-block submatrixes, $\mathcal{X}^{(1)}$ (i.e., $k = 1$), is the collection of matrixes $\mathbf{X}[R_1, C_1], \dots, \mathbf{X}[R_r, C_r]$. The collection of r-block submatrices, $\mathbf{X}^{(r)}$ (i.e., $k = r$), includes only the original matrix $\mathbf{X}$. For the example shown in Figure 4, $\mathcal{X}^{(1)}$ includes $\mathbf{X}[R_1, C_1]$ and $\mathbf{X}[R_2, C_2]$, and $\mathcal{X}^{(2)}$ includes only $\mathbf{X}[R_1 \cup R_2, C_1 \cup C_2]$, which is the same as $\mathbf{X}$.

As implied in the definition of M, the algorithm that calculates M will compute d^* first and then compute M as $n - d^*$. The algorithm starts by testing the rank of a design matrix while eliminating a certain number of rows from it until d, the number of eliminated rows, reaches a bound. The following theorem (from Cho, Chen, and Ding 2007b, but rewritten in matrix notation) specifies the bound for switching. The notation $d^*(\mathbf{X}[I, J])$ denotes the d^* of submatrix $\mathbf{X}[I, J]$.

Theorem 1. For any $k \in \{1, 2, \dots, r\}$, if $d^*(\mathbf{X}) \geq \frac{k+1}{k}|S| - 1$, then

$$d^*(\mathbf{X}) = \min_U \left\{ d^*\left(\mathbf{X}\left[\bigcup_{i \in U} R_i, \bigcup_{j \in U} C_j \right] \right) : U \subseteq \{1, 2, \dots, r\} \text{ and } |U| = k \right\}.$$

Theorem 1 says that $d^*(\mathbf{X})$ can be obtained from d^*'s of submatrixes, provided that $d^*(\mathbf{X}) \geq \frac{k+1}{k}|S| - 1$. Note that Theorem 1 holds for any $k \in \{1, 2, \dots, r\}$ and that the minimization is over U. After d^* is obtained, M is simply $n - d^*$.

The bound-and-decompose algorithm based on Theorem 1 is shown in a flowchart in Figure 6.

In the flowchart, we begin with $d = 1$ and increase d by 1 at each iteration, with the knowledge that $d^* \geq$ current d. The dashed line in the middle divides the algorithm flow into two segments. The first segment is for the rank testing of the original matrix, the same as for exhaustive rank testing. The second segment is for the rank testing of submatrixes.

Theorem 1 also suggests that the value of the bound allowing testing of the ranks of submatrixes depends on what combinations of submatrixes are to be tested, because the bound condition $\frac{k+1}{k}|S| - 1$ contains the block-combination parameter k. When testing one-block submatrixes (i.e., $k = 1$), the bound on d is $d \geq 2|S| - 1$, meaning that the rank of the original matrix should be tested until $d = 2|S| - 2$, after which one-block submatrix rank-testing can be done. For testing the rank of $(r - 1)$-block submatrixes (i.e., $k = r - 1$), the bound on d is $d \geq \frac{r}{r-1}|S| - 1$, meaning the original matrix should be tested until $d = \lceil \frac{r}{r-1}|S| \rceil - 2$, where $\lceil a \rceil$ is the smallest integer $\geq a$, after which $(r - 1)$-block submatrix testing can be done. Evidently, the bound for the $(r - 1)$-block submatrix testing is not greater than that for the one-block testing for any $r > 2$ and $|S| > 1$. In Theorem 1, although k takes values up to r, the bound for r-block submatrix testing does not differ from that for rudimentary exhaustive rank testing, because the r-block submatrix is the same as the original matrix. On the other hand, when $|S| = 0$ or 1, the condition stated in Theorem 1 always holds, so one-block submatrix testing can be done without first testing the original matrix. In other words, there is no bound for d when $|S| = 0$ or $|S| = 1$.

As such, when a matrix decomposition results in r submatrixes, $\mathbf{X}[R_1, C_1], \dots, \mathbf{X}[R_r, C_r]$, the smallest bound on d allowing for testing a submatrix is $\frac{r}{r-1}|S| - 1$, that for the $(r - 1)$-block testing. It then becomes clear that our algorithm runs the exhaustive rank testing for d from 1 up to $\lceil \frac{r}{r-1}|S| \rceil - 2$ and then switches to the submatrix testing. As d increases, rank testing for submatrixes with smaller blocks may be allowed; however, when $d \geq \min_j(|R_j| - |C_j|) + 1$, M is simply $n - \min_j(|R_j| - |C_j|) - 1$, precluding the need for further rank testing. That is why we verify that d is $> \min_j(|R_j| - |C_j|) - 1$ at the beginning of each iteration. If it is not, then the operations shown in the upper part of the flowchart are simple, including comparing d with the bound condition and, if d is smaller than the bound, performing the same action as for exhaustive rank testing.

In the bottom part of the flowchart in Figure 6, for a given d, there are multiple possibilities for rank testing of the k-block submatrixes. The approach involving the least computation time should be chosen. The computation time for rank testing depends on the number of possible reduced matrixes, $\mathbf{X}_{(-d)}$, that can be generated by removing d rows from $\mathbf{X}$. Let $N_{(-d)}(k)$ denote the total number of reduced matrixes generated from *all* of the submatrixes in $\mathcal{X}^{(k)}$. For a given d, k^* should be chosen that minimizes $N_{(-d)}(k)$, namely,

$$k^* = \arg\min_k N_{(-d)}(k),$$

and then the ranks of the resulting reduced matrixes, $\mathbf{X}_{(-d)}^{(k^*)}$, should be tested for $\mathbf{X}^{(k^*)} \in \mathcal{X}^{(k^*)}$.

Suppose that we apply this algorithm to the simple example shown in Figure 4, following the procedure illustrated in

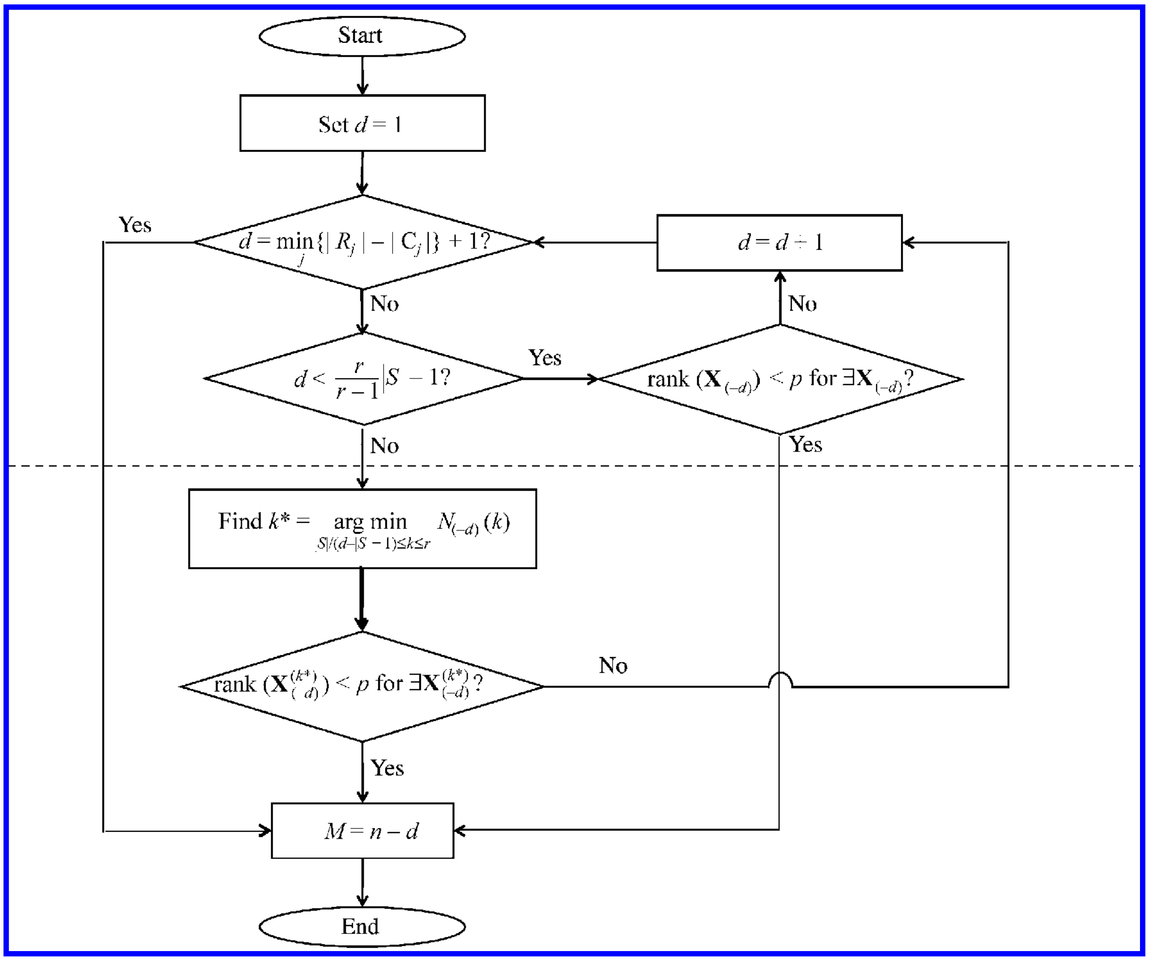

Start
Set $d = 1$
Yes
$d = \min_j\{|R_j| - |C_j|\} + 1$?
No
$d = d + 1$
No
$d < \frac{r}{r-1}|S| - 1$?
Yes
rank $(\mathbf{X}_{(-d)}) < p$ for $\exists \mathbf{X}_{(-d)}$?
No
Yes
Find $k^* = \arg\min_{|S|/(d-|S|-1) \le k \le r} N_{(-d)}(k)$
rank $(\mathbf{X}^{(k^*)}_{(-d)}) < p$ for $\exists \mathbf{X}^{(k^*)}_{(-d)}$?
No
Yes
$M = n - d$
End

Figure 6. Bound-and-decompose algorithm.

Figure 6. Based on the matrix decomposition described in Section 2.1, we know that the design matrix has two blocks and one border row, meaning that $r = 2$ and $|S| = 1$. The decomposed submatrixes are $\mathbf{X}[R_1, C_1]$ and $\mathbf{X}[R_2, C_2]$, where $R_1 = \{1, 4, 6\}$ and $C_1 = \{4\}$ and $R_2 = \{2, 3, 5, 6\}$ and $C_2 = \{1, 2, 3\}$, so that $|R_1| = 3$, $|R_2| = 4$, $|C_1| = 1$, $|C_2| = 3$, and $\min_j(|R_j| - |C_j|) + 1 = 2$.

The algorithm starts with $d = 1$. Because $d = 1$ is less than $\min_j(|R_j| - |C_j|) + 1$, the algorithm takes the "No" branch. Then we can verify that $d = 1$ is not strictly less than $\frac{r}{r-1}|S| - 1$ (which is 1 here). Thus the algorithm takes the "No" branch again and moves to the next step, determining k^*. Here k can take only one of two values: 1 or 2. When $k = 1$, $\mathcal{X}^{(1)}$ contains two submatrixes, 3×1 $\mathbf{X}[R_1, C_1]$ and 4×3 $\mathbf{X}[R_2, C_2]$. For $d = 1$ and $k = 1$, the total number of reduced matrixes that can be generated from $\mathcal{X}^{(1)}$ is $3 + 4 = 7$, so $N_{(-1)}(1) = 7$. When $k = 2$, $\mathcal{X}^{(2)}$ contains only one matrix, the 6×4 $\mathbf{X}[R_1 \cup R_2, C_1 \cup C_2]$, which is the same as $\mathbf{X}$. For $d = 1$ and $k = 2$, the total number of reduced matrixes that can be generated from $\mathcal{X}^{(2)}$ is 6, so $N_{(-1)}(2) = 6$. This gives $k^* = 2$, meaning that the original matrix $\mathbf{X}$ is used for the rank testing. In the next step, six reduced submatrixes, $\mathbf{X}_{(-1)}$'s (after each one of the six rows is removed), are tested, but none is found to be rank-deficient; thus the algorithm takes the "No" branch, increases d to 2, and then determines whether the new d is the same as $\min_j(|R_j| - |C_j|) + 1$. This time, it is. Thus the algorithm stops at $d = 2$, implying that $d^* = 2$. Subsequently, M can be computed as $M = n - d^* = 6 - 2 = 4$.

For this simple problem, we did not use the decomposed submatrixes to determine M, because we tested only the case when $d = 1$; in this case, whether the whole matrix or the submatrixes are tested makes little difference. For more complicated problems involving larger values of d, matrix decomposition will greatly benefit the computation, as shown in the example presented in Section 4.

2.3 Applicability of the Bound-and-Decompose Method

A structured matrix may not contain sufficient number of zero elements to qualify as a sparse matrix. Consider, for example, a structured matrix $\mathbf{X}$ such as

$$\mathbf{X}^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}. \qquad (7)$$

This $\mathbf{X}$ is structured, meaning that there are linearly dependent rows in $\mathbf{X}$, but it is not sparse. When directly applying the procedure described in Section 2.1, we end up with rows $\{3, 4, 5, 6, 7\}$ as the border rows, plus one block $\mathbf{X}[\{1, 2\}, \{1, 2\}]$. Apparently, this decomposition is not very useful for the subsequent bound-and-decomposition procedure.

One way to transform $\mathbf{X}$ into a sparse matrix without changing its M values is to use the reduced row echelon form (RREF) of $\mathbf{X}$. (See the Appendix for a definition of the RREF.) Theorem 2 guarantees that using the RREF of a matrix will not alter the value of M.

Theorem 2. Denote by $\mathbf{W}$ the transposed RREF of $\mathbf{X}^T$. Then

$$M(\mathbf{W}) = M(\mathbf{X}).$$

The proof of Theorem 2 necessarily involves matroid theory (see Oxley 1992, pp. 80–81, for a proof).

Returning to the matrix in (7), we obtain $\mathbf{W}^T = RREF(\mathbf{X}^T)$ such as

$$\mathbf{W}^T = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0.5 \end{pmatrix}.$$

Then we have a sparse matrix $\mathbf{W}$ and can obtain a useful BBDF for the bound-and-decomposition algorithm. (This $\mathbf{W}^T$ itself is in a BBDF.)

Unfortunately, the RREF of a structured matrix is not always sufficiently sparse to confer a significant computational benefit to the bound-and-decompose algorithm. This raises the question of applicability or "usefulness" conditions for our method. To have a significant computational benefit, the following conditions must be satisfied:

- $\mathbf{X}$ or the RREF of its transpose can be transformed to BBDF with $r > 1$.
- The maximum number of columns in any $\mathbf{B}_i$ submatrix is much less than p.
- The number of rows in $(\mathbf{S}_1\ \mathbf{S}_2\ \cdots\ \mathbf{S}_r)$ is much less than n.

Another question is what type of structured linear models encountered in practice are likely to meet these conditions. Our experience suggests that engineering systems, especially those comprising subsystems linked through an interconnecting element, are likely to have a "useful" (according to the aforementioned "usefulness" conditions) BBDF design matrix. Row and column exchange operations may be needed to permute a model matrix with hidden bordered block structures, as shown in Figure 2(c), into an explicit BBDF; this can be done by using the graph-based procedure described in Section 2.1. Each $\mathbf{B}_i$ submatrix in the BBDF corresponds to the unique measurements and of an individual subsystem, and the $[\mathbf{S}_1\ \mathbf{S}_2\ \cdots\ \mathbf{S}_r]$ border matrix corresponds to measurements that interconnect the subsystems.

The practical implications of the "usefulness" conditions are that the number of subsystems should be more than 1, each of the subsystems should be much smaller than the whole system, and the number of observations in the interconnecting element should be much lower than the total observations n. The multistation assembly process introduced in Section 1 is an example of this. In that process, the sensors placed at Stations 1 and 2 can measure locator deviations only at their respective stations, making these the subsystems, and the two sensors at Station 3 are the interconnecting elements, the measurements of which are affected by locator deviations at all three stations. There are a total of $r = 4$ subsystems, each of which is roughly one-quarter the size of the whole system. The number of observations in the interconnecting element is 2, much smaller than $n = 26$.

Other suitable applications that we have encountered include clustered wireless sensor networks (Cho et al. 2007a) and the state estimation problem in electrical power systems (Alvarado and Tinney 1990; Mili, Phaniraj, and Rousseeuw 1990; Mili and Coakley 1996). When studying a calibration problem for a clustered sensor network, Cho et al. (2007a) found that the resulting design matrix appeared to be sparse and could be transformed into a useful BBDF, with the blocks corresponding to individual sensor clusters and the border rows corresponding to the set of the between-cluster links. For the electrical power systems, this consists of a number of interrelated branches. The whole system could be large, with tens or hundreds of states to be estimated. According to Alvarado and Tinney (1990), Mili, Phaniraj, and Rousseeuw (1990), Mili and Coakley (1996), design matrixes in their applications were very sparse, because most of the measurements were related to only a handful of states, and the number of nonzero elements in a design matrix ranged from 5% to 10%. We expect that this sort of design matrix can be transformed into a useful BBDF, with the blocks corresponding to individual branches and the border rows corresponding to the interconnecting links.

Along with engineering models, general regression problems also can have sparse model matrixes. Mili and Coakley (1996) noted that "*large-scale regression models*" are likely to have sparse design matrixes. To be more specific, consider a regression problem based on text data. Each predictor may correspond to a word, with $x_{ij} = 1$ if the jth word occurs in the ith document and 0 otherwise. Given a large collection of documents, if the words appear in only a small number of documents, then the resulting $\mathbf{X}$ matrix will be sparse. Generally, a common source of sparseness in regression model matrixes is the inclusion of categorical variables. These categorical variables must be coded using a set of dummy variables. For example, suppose that we have a four-level categorical variable; that is, the variable can take one of four permissible values. We can use the following dummy variables to represent the four levels: (1 0 0) for level 1, (0 1 0) for level 2, (0 0 1) for level 3, and $(-1\ -1\ -1)$ for level 4. This specific coding scheme is known as *effect code* in categorical variable regression problems. If dummy variables are used to represent the categorical variables, then the model matrixes will contain many 0's and thus will be sparse.

3. MORE ABOUT THE BREAKDOWN POINT OF LTS ESTIMATION

Section 2 provides a computational procedure for the quantity M, which is important for assessing the breakdown point as well as determining the trimming parameter h for an LTS estimator. When h is chosen such that $h_L \le h \le h_U$, the resulting LTS estimator is guaranteed to attain the maximum breakdown point $\epsilon^*_{\max,n}$ in Equation (4). A question naturally follows: What happens if an h outside the maximum breakdown range, namely $h < h_L$ or $h > h_U$, is used?

Because h is the number of data points used for estimation, a higher breakdown point may be expected when using a smaller h, and a lower breakdown point may be expected when using a larger h. It is in fact true that using $h > h_U$ results in a lower breakdown point than $\epsilon^*_{\max,n}$, as formally stated in theorem 5.2 of Mili and Coakley (1996).

Theorem 3 [From theorem 5.2 of Mili and Coakley (1996)]. An LTS estimator satisfying $h \geq h_U$ has a breakdown point given by $(n - h + 1)/n$.

When $M \approx p - 1$ [then $h_U \approx (n + p)/2$], using $h > h_U$ is sometimes a practical choice. From Theorem 3, the resulting breakdown point of using $h > h_U$ can be estimated. If working with relatively clean data with only a few outliers, then trimming almost half the observations is unnecessary. Even though using $h > h_U$ could result in a lower breakdown point, depending on circumstances, such a choice could be sound and preferable. Choosing a good value of h necessarily involves other considerations, including the expected frequency of outliers.

Using $h < h_L$ means that more measurements are trimmed off. Doing this does not guarantee that the resulting LTS estimator can achieve the maximum breakdown point as achieved by using the h in the maximum breakdown range. On the other hand, using $h < h_L$ can be beneficial when considering the possibility of breakdown of such an LTS estimator. An LTS estimator with h slightly smaller than h_L may break down only in the presence of outliers to the y values corresponding to a set of "worst-case" rows of $\mathbf{X}$, but this estimator still could provide robustness to a larger number of outliers (larger than the maximum breakdown point) falling in other rows. The set of "worst-case" rows of $\mathbf{X}$ is defined as the smallest row set whose removal will make the resulting $\mathbf{X}$ rank-deficient; we give an example of this later. If the likelihood of outliers falling in the "worst-case" row is small (as might be expected when $M \gg p - 1$), then in practice settings, using a smaller h could provide better overall robustness than using $h_L \leq h \leq h_U$.

To illustrate this, consider a design matrix with dimension 100×3 (namely $n = 100; p = 3$),

$$\mathbf{X}^T = \begin{pmatrix} 1 & 1 & \cdots & 1 & 0 & 0 & 0 \\ 1 & 2 & \cdots & 97 & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 & 1 \end{pmatrix}. \qquad (8)$$

Given this matrix's relatively simple structure, it is straightforward to determine that $M = 97$, $h_U = h_L = 99$, and $\epsilon^*_{\max,n} = 2/100$. Thus the maximum breakdown choice of h is 99, meaning that the LTS estimator with $h = 99$ can tolerate one outlier regardless of the row in which the outlier occurs. But this LTS estimator will always break down when more than one outlier exists. Suppose that we use $h = 98$, a value smaller than h_L. Even though two measurements are trimmed off, the LTS still could break down with the presence of two outliers. Recall that, intuitively, the robust estimation mechanism involves using most of the observations to protect against the effects of outliers. Bearing this in mind, it is not difficult to see that the LTS estimator with $h = 98$ has a high likelihood of breaking down only when both outliers occur in rows $\{98, 99, 100\}$, whereas the LTS estimator will not break down when the two outliers occur in rows $\{1, \ldots, 97\}$ and is much less likely to break down when one outlier occurs in rows $\{1, \ldots, 97\}$ and the other occurs in rows $\{98, 99, 100\}$. For this reason, we call rows $\{98, 99, 100\}$ the "worst-case" rows. This is consistent with the definition of "worst-case" rows, because $\{98, 99, 100\}$ is the smallest row set whose removal from $\mathbf{X}$ will cause $\mathbf{X}$ to be singular. If we assume that the outlier(s) can occur independently of all of the rows with equal probability, then the likelihood of both outliers occurring in rows $\{98, 99, 100\}$ is rather small, meaning that the LTS estimator with $h = 98$ has a low likelihood of breakdown.

Using the LTS estimator with $h = 98$ entails an additional risk for not being able to tolerate a single outlier, a problem not posed by the LTS estimator using the maximum breakdown h. To see this, suppose that y_{98} is an outlier; then trimming off y_{99} and y_{100} and setting $\hat{\beta}_3 = y_{98}$ will always lead to an exact fit of y_3, while producing good fits of the other observations. Thus, if y_{98} is a single outlier and $h = 98$, then it is *possible* that $\{99, 100\}$ will be trimmed off, causing a breakdown. But again, the likelihood of this is rather low, especially when n is large.

To get a more quantitative sense of the likelihood of breakdowns as mentioned earlier, we ran a simulation using an LTS estimation with $h = 98$ and the matrix in (8). We used $\boldsymbol{\beta} = [0 \quad 0 \quad 0]^T$ with a set of n random errors following N(0, 1) to generate $\mathbf{y}$, and added a large magnitude of 15 to y_{98} as the outlier. We performed 10,000 LTS regressions by using $h = 98$. The percentage of trimming off rows 99 and 100 (i.e., y_{98} stays and the LTS estimator breaks down) was 1.0%. Furthermore, keeping the first outlier at row 98, we added the second outlier at a randomly chosen row. Out of 10,000 simulations, 224 runs had the second outlier in the last two rows (either y_{99} or y_{100}), and 9,776 runs had the second outlier in the first 97 rows. When the second outlier was in the last two rows, the likelihood of retaining both outliers (namely the good observation among the last three rows is trimmed off) was 98.2%. But when the second outlier was not in the last two rows, both outliers were always correctly trimmed off in the simulations. Overall, the likelihood of including both outliers (so the estimation breaks down) was 2.2%. This relatively low overall percentage suggests that the LTS estimation using $h = 98$ provides fairly robust estimation for the matrix $\mathbf{X}$ in (8). In general, for a case with a good possibility that more than $n - h_L$ outliers are present, one certainly should not be bound by the maximum breakdown h when choosing the trimming parameter; a smaller h could confer an extra degree of robustness.

4. REVISIT THE ASSEMBLY PROCESS EXAMPLE

The Design Matrix

Here we revisit the assembly example in Figure 1 and use it as an illustrative example. The design matrix $\mathbf{X}$ for this system is shown in Figure 7.

Calculating *M*

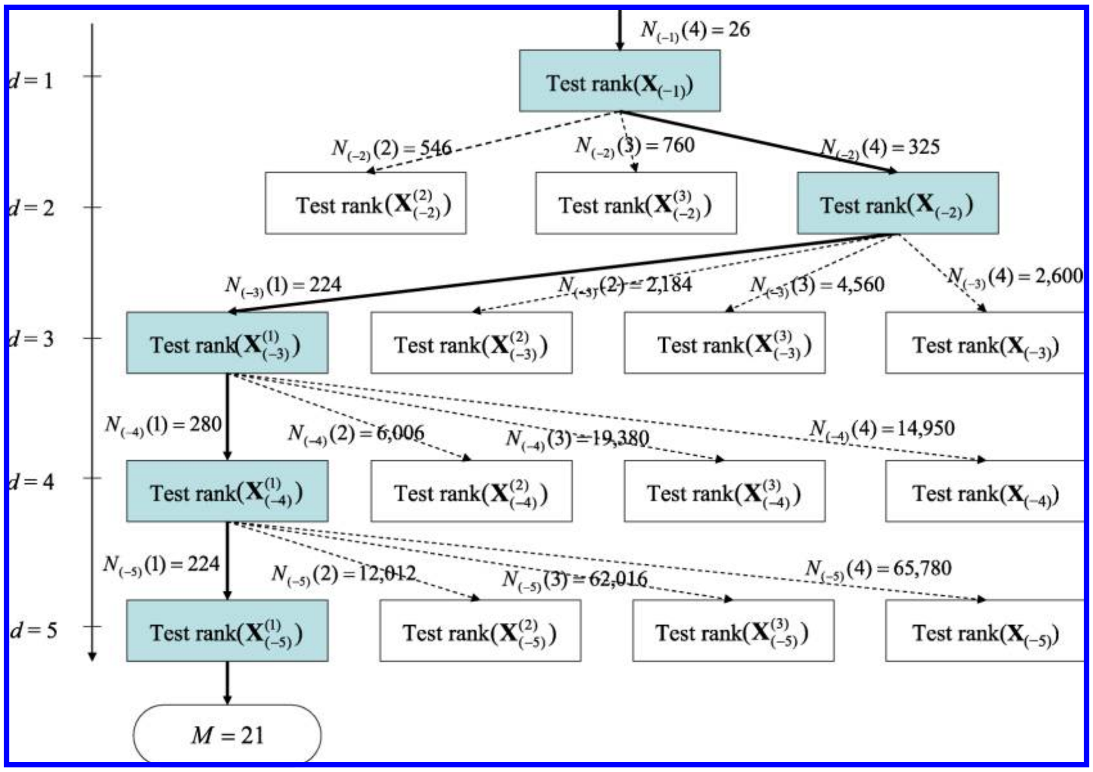
First, we use the bigraph-based method to transform $\mathbf{X}$ into a BBDF. For the $\mathbf{X}$ in Figure 7, $R = \text{Row}(\mathbf{X}) = \{1, 2, \ldots, 26\}$ and $C = \text{Col}(\mathbf{X}) = \{1, 2, \ldots, 12\}$. Using the bipartite graph procedure described in Section 2.1, we find that $S = \{10, 14\}$. Then we remove S and decompose the reduced bipartite graph associated with $\mathbf{X}[R - S, C]$. Finally, we obtain $C_1 = \{1, 2, 3\}$, $C_2 = \{4, 5, 6\}$, $C_3 = \{7, 8, 9\}$, and $C_4 = \{10, 11, 12\}$. Hence, $r = 4$ and $\frac{r}{r-1}|S| = 2.67$.

Second, we use the decomposition algorithm given in Section 2.2 to find M; the computation procedure is illustrated in Figure 8. Given $\frac{r}{r-1}|S| = 2.67$, the bound on d that allows us

$$\mathrm{X}=\begin{pmatrix}
1 & 1.8 & -1.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 1.8 & -1.8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & .2857 & -.2857 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & .2857 & -.2857 & 0 & 0 & 0\\
1 & -.9 & .9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1.26 & -.26 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & .56 & .44 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & .55 & -.55 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & -.35 & .35 & 0 & 0 & 0 & 0 & 0 & 0\\
-1 & -.3313 & 0 & 1 & .55 & -.2186 & -1.0041 & -.0297 & 0 & .0041 & 0 & .0297\\
0 & 0 & 0 & 0 & 1.09 & -.09 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .5 & .5\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -.375 & 1.375\\
-1 & .2108 & 0 & 1 & -.35 & .1392 & -.9675 & .2380 & 0 & -.0325 & 0 & -.2380\\
0 & 0 & 0 & 0 & -.16 & 1.16 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 1 & -.2857 & .2857 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.2857 & -.2857 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .5714 & .4286 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -.1429 & 1.1429 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2.25 & -2.25\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 & -2\\
0 & 0 & 0 & 0 & .465 & .535 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & -.14 & 1.14 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 1 & .55 & -.55 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1.125 & 1.125\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1.375 & -.375
\end{pmatrix}
\begin{array}{l}1\\2\\3\\4\\5\\6\\7\\8\\9\\10\\11\\12\\13\\14\\15\\16\\17\\18\\19\\20\\21\\22\\23\\24\\25\\26\end{array}$$

Figure 7. The design matrix of the multistation assembly process. The numbers outside the bracket on the right are the row numbers.

to switch to submatrix testing is $\lceil \frac{r}{r-1}|S|\rceil - 1 = 2$. Thus we run the rank testing on $\mathbf{X}$ for $d = 1$, and because $\mathbf{X}_{(-1)}$ is of full column rank, we conclude that $M < 25$. Starting with $d = 2$, we can test the k-block submatrixes. When $d = 2$, we may test the 2-block, 3-block, and 4-block submatrixes, and it turns out that $N_{(-2)}(4)$ is the smallest of the three alternatives. From Fig-



$N_{(-1)}(4)=26$
$d=1$
Test rank($\mathbf{X}_{(-1)}$)
$N_{(-2)}(2)=546$
$N_{(-2)}(3)=760$
$N_{(-2)}(4)=325$
$d=2$
Test rank($\mathbf{X}^{(2)}_{(-2)}$)
Test rank($\mathbf{X}^{(3)}_{(-2)}$)
Test rank($\mathbf{X}_{(-2)}$)
$N_{(-3)}(1)=224$
$N_{(-3)}(2)=2{,}184$
$N_{(-3)}(3)=4{,}560$
$N_{(-3)}(4)=2{,}600$
$d=3$
Test rank($\mathbf{X}^{(1)}_{(-3)}$)
Test rank($\mathbf{X}^{(2)}_{(-3)}$)
Test rank($\mathbf{X}^{(3)}_{(-3)}$)
Test rank($\mathbf{X}_{(-3)}$)
$N_{(-4)}(1)=280$
$N_{(-4)}(2)=6{,}006$
$N_{(-4)}(3)=19{,}380$
$N_{(-4)}(4)=14{,}950$
$d=4$
Test rank($\mathbf{X}^{(1)}_{(-4)}$)
Test rank($\mathbf{X}^{(2)}_{(-4)}$)
Test rank($\mathbf{X}^{(3)}_{(-4)}$)
Test rank($\mathbf{X}_{(-4)}$)
$N_{(-5)}(1)=224$
$N_{(-5)}(2)=12{,}012$
$N_{(-5)}(3)=62{,}016$
$N_{(-5)}(4)=65{,}780$
$d=5$
Test rank($\mathbf{X}^{(1)}_{(-5)}$)
Test rank($\mathbf{X}^{(2)}_{(-5)}$)
Test rank($\mathbf{X}^{(3)}_{(-5)}$)
Test rank($\mathbf{X}_{(-5)}$)
$M=21$

Figure 8. Execution of the decomposition algorithm for the multistation assembly example.

Table 1. Computation time for M by exhaustive search and bound & decomposition

Design matrix				Computation time	
Dimension	$\lvert S\rvert$	r	M	Exhaustive search	Bound & decompose
26×12	2	4	21	8 sec	0.1 sec
67×24	3	8	59	>120 h	6.1 min
222×55	2	11	207	>120 h	16.5 min
1009×252	1	41	993	>120 h	38.2 min

ure 8, $N_{(-2)}(2) = 6 \times C_{14}^2 = 546$, $N_{(-2)}(3) = 4 \times C_{20}^2 = 760$, and $N_{(-2)}(4) = C_{26}^2 = 325$. Thus $k^* = 4$ for $d = 2$, and we test the ranks of $\mathbf{X}_{(-2)}$. Again, we find that $M < 24$. We repeat the procedure for $d = 3, 4, 5$. In those cases, $k^* = 1$. Eventually, we test the ranks of submatrixes $\mathbf{X}_{(-5)}^{(1)}$ for $\mathbf{X}^{(1)} \in \mathcal{X}^{(1)}$ and find that $d^* = 5$, so $M = 21$.

The computational path is highlighted by the solid arrows in Figure 8, where the dashed line indicates other possible alternatives. Adding the numbers on the solid arrows gives the number of the maximum iterations used by the algorithm: 1079. In comparison, had we used the exhaustive rank testing on $\mathbf{X}$ entirely, then the iteration number could have been as large as 83,681, which actually equals the total of the numbers of the rightmost arrows. The algorithm, coded in MATLAB, goes through 904 actual iterations in 0.1 second, whereas the exhaustive rank testing takes 8 seconds to complete 45,565 actual iterations. A larger $\mathbf{X}$ can result in greater potential savings. We also created larger systems with design matrixes 67×24, 222×55, and 1009×252. The computational results are summarized in Table 1, where ">120 hours" is reported for the three large systems, because we manually stopped the exhaustive rank testing procedure when it had run for 120 hours (i.e., 5 days).

The LTS Estimator

Using the assembly process ($n = 26$, $p = 12$, and $M = 21$), we illustrate the robustness of an LTS estimator with different trimming parameters in the presence of outlier(s). We show the performance only in the "worst-case" application. In this application, we identify a set of "worst-case" rows as $\{13, 20, 21, 25, 26\}$ because this is one of the smallest row sets whose removal will cause the design matrix to be rank-deficient. In what follows, the outliers are added to the rows selected from this "worst-case" row set.

We include three choices for the trimming parameter:

1. Given $M = 21$, we have $h_L = h_U = 24$, so that the h recommended by Mili and Coakley (1996) is 24. This suggests that the LTS with $h = 24$ can tolerate up to $n \cdot \epsilon_{\max,n} = 2$ outliers.
2. The trimming parameter using the formula of Rousseeuw (1984), which is designed for the general position, is $h = [n/2] + [p/2] = 19$.
3. We also choose $h = 22$ as an intermediate value.

We simulated $K = 1000$ observations using the following parameters: $\mathbf{e}$ and $\boldsymbol{\beta}$ generated following a normal distribution with mean 0 and standard deviation 0.1, and an outlier simulated by adding a large value of magnitude 50 to the y value corresponding to a fixed row.

We consider the scenarios of zero, one, two, and three outliers. Outliers were added to the following rows: y_{26} for one outlier, $\{y_{25}, y_{26}\}$ for two outliers, and $\{y_{13}, y_{25}, y_{26}\}$ for three outliers.

We wished to determine how often an LTS estimator broke down, with a breakdown counted when at least one outlier was not trimmed off correctly. We also computed the mean squared error (MSE) as follows:

$$\text{MSE} = \frac{1}{K}\sum_{t=1}^{K}(\boldsymbol{\beta}_t - \hat{\boldsymbol{\beta}}_t)^T(\boldsymbol{\beta}_t - \hat{\boldsymbol{\beta}}_t).$$

Table 2 summarizes the number of breakdowns and the corresponding MSE values, given different h's and numbers of outliers.

Table 2 shows that the LTS ($h = 24$) never breaks down when the number of outliers is less than or equal to two, but always breaks down when there are more than two outliers. This is as expected based on the theoretical results given by Mili and Coakley (1996). The likelihood of breakdown of the LTS ($h = 19$ or $h = 22$) is not high when only one outlier is present, but it increases substantially when two outliers are present (10% to 20%). Bear in mind that the outliers are added to the "worst-case" rows in this study. Had we added the outliers randomly, fewer breakdowns would be expected. When there are more than two outliers, the LTS with $h < h_L$ outperforms the LTS ($h = 24$) by breaking down far fewer times, but in this case, the LTS ($h = 19$) does not perform as well as the LTS ($h = 22$) under all scenarios.

The absolute magnitude of the MSEs is not very significant, because the large MSE values given in Table 2 are clearly a result of using outliers of large magnitude. Note that the values in parentheses after an MSE value represent the MSE when no breakdown occurs and the MSE when breakdown does occur; for example, 648.6 (1.27, 3,143) indicates that the MSE for LTS ($h = 19$) with the presence of two outliers is 648.6, whereas the MSE for the 794 instances when no breakdown occurs is 1.27, and the MSE for the 206 instances when breakdown does occur is 3,143.

In the absence of outliers, an LS estimator outperforms an LTS estimator because the latter uses only a subset of sensor measurements, so that its efficiency suffers. With the presence of outliers, the increase in MSE value depends on how often an estimator breaks down, how many outliers are included in the resulting estimators, and how many "good" observations are trimmed off. With no breakdown at one and two outliers, the LTS ($h = 24$) produces an MSE comparable to that of no outlier, whereas the MSEs of the other estimators increase substantially.

Table 2. Number of breakdowns and MSEs of LS estimator and LTS ($h = 19, 22, 24$) estimators. The number in parentheses after the number of breakdowns represent how many times an LTS estimator includes one, two, or three outliers; for example, 281 (151, 130, 0) indicates 281 total breakdowns, 151 including only one outlier, 130 including two outliers, and 0 including three outliers. The value in parentheses after the MSE value represents the MSE when there is no breakdown and when there is breakdown; for example, 648.6 (1.27, 3,143) indicates that the MSE for LTS ($h = 19$) with the presence of two outliers is 648.6, the MSE for the 794 instances when no breakdown occurred is 1.27, and the MSE for the 206 instances when breakdown occurred is 3,143

Number of outliers		LS	LTS ($h = 19$)	LTS ($h = 22$)	LTS ($h = 24$)
0	# breakdowns	0	0	0	0
	MSE	0.048	0.872	0.241	0.097
1	# breakdowns	1000	25	4	0
	MSE	588.9	205.0 (0.481, 8,182)	23.40 (0.275, 5,780)	0.106
2	# breakdowns	1000	206 (199, 7)	95 (94, 1)	0
	MSE	1164	648.6 (1.27, 3,143)	240.2 (1.01, 2519)	0.210
3	# breakdowns	1000	281 (151, 130, 0)	125 (64, 61, 0)	1000 (1000, 0, 0)
	MSE	3,123	1.86×10^5 ($1.63, 6.63 \times 10^5$)	6.89×10^4 ($1.59, 5.51 \times 10^5$)	1197

With the presence of three outliers, the MSE of LTS estimator $h = 19$ or $h = 22$ is larger than that of LTS estimator $h = 24$. This difference is rooted in the manner in which the outliers, as well as the good observations, are trimmed. With $h < h_L$ (as is the case for $h = 22$ and $h = 19$), the trimming occurs in a highly unpredictable fashion. Depending on what is trimmed and what remains, certain fits can have a very large sum of squared errors (SSE), because the outliers dominate the fit. For example, in one of the runs in which rows $\{12, 21, 25, 26\}$ are trimmed, the resulting SSE of the fit is 3.87×10^6; however, in another run in which rows $\{12, 13, 21, 25\}$ are trimmed, the resulting SSE is 6.04×10^3. As such, those trimmings resulting in a very large SSE eventually contribute to the high MSE value. When $h = 24$, the situation is different. With h in the maximum breakdown range, LTS is able to always trim off two (out of three) outliers without removing any good observations. For all of the simulation runs, LTS produces consistent fits with SSEs on the order of 10^3. The action of trimming off good observations while retaining outliers produces an undesirable escalation in MSE values, so that the LTS with $h = 19$ or $h = 22$ produces an even greater MSE than the LS estimator with three outliers; the LS estimator can be considered as an LTS with $h = 26$, but, on the other hand, does not trim off any "good" observations.

5. CONCLUSION

In this articles we have presented an efficient algorithm for computing the maximum breakdown point for sparse linear regressions. The output of the algorithm can aid assessment of the robustness level of a regression estimator and guide the choice of trimming parameter to use in the LTS estimator. We expect that our methods will be useful in regression problems involving categorical variables or engineering systems consisting of many interconnected subsystems.

Through our research, we also have discovered that even though the trimming parameter chosen from the maximum breakdown range can guarantee the promised robustness in estimation, using a trimming parameter lying outside that range can have a benefit as well. In particular, using a smaller trimming parameter can confer an extra degree of robustness in cases when a breakdown is not likely.

A natural follow-up to the present analysis is to investigate how the design can be improved to increase the breakdown point. We believe that the relationships between the potential design variables and the breakdown point are not straightforward. Sophisticated optimization algorithms need to be devised for this purpose. On the other hand, the following intuition from our understanding may be useful for design. A design matrix with balanced block sizes will be beneficial, because a very small block will keep the breakdown point low for the whole system even if the other blocks are large. Because block size is related to the size of subsystems in physical settings, this suggests that designing size-balanced subsystems will benefit the system's estimation robustness.

ACKNOWLEDGMENTS

Financial support was provided by National Science Foundation Grants CMMI-0348150, CMMI-0528735, and CMMI-0726939/0727305. The authors thank the editors and the referees for their valuable comments and suggestions.

APPENDIX: DEFINITION OF RREF (LAY 1997)

A matrix is in RREF if it satisfies the following requirements:

- All nonzero rows are above any rows of all 0's.
- The leading coefficient of a row is always to the right of the leading coefficient of the row above it.
- All leading coefficients are 1.
- All entries above a leading coefficient in the same column are 0.

A matrix can be reduced to RREF using elementary row operations.

[Received June 2006. Revised August 2008.]

REFERENCES

Alvarado, F. L., and Tinney, W. F. (1990), "State Estimation Using Augmented Blocked Matrices," *IEEE Transactions on Power Systems*, 5, 911–921.

Aykanat, C., Pinar, A., and Çatalyürek, Ü. V. (2004), "Permuting Sparse Rectangular Matrices Into Block-Diagonal Form," *SIAM Journal on Science Computing*, 25, 1860–1879.

Cho, J., Ding, Y., Chen, Y., and Tang, J. (2007a), "Robust Calibration for Localization in Clustered Wireless Sensor Networks," in *Proceedings of the 2007 IEEE Conference on Automation Science and Engineering*, Scottsdale, AZ.

Cho, J. J., Chen, Y., and Ding, Y. (2007b), "On the (Co)Girth of Connected Matroids," *Discrete Applied Mathematics*, 155, 2456–2470.

Coakley, C. W., and Mili, L. (1993), "Exact Fit Points Under Simple Regression With Replication," *Statistics & Probability Letters*, 17, 265–271.

Davies, P. L. (1993), "Aspects of Robust Linear Regression," *Annals of Statistics*, 21, 1843–1899.

Donoho, D. L., and Huber, P. J. (1983), "The Notion of Breakdown Point," in *A Festschrift for Erich L. Lehman*, eds. P. Bickel, K. Doksum, and J. L. Hodeges Jr., Belmont, CA: Wadsworth, pp. 157–184.

Even, S. (1979), *Graph Algorithms*, Potomac, MD: Computer Science Press.

Ferris, M. C., and Horn, J. D. (1998), "Partitioning Mathematical Programs for Parallel Solution," *Mathematical Programming*, 80, 35–61.

Jin, J., and Shi, J. (1999), "State–Space Modeling of Sheet Metal Assembly for Dimensional Control," *ASME Transactions*, *Journal of Manufacturing Science and Engineering*, 121 (4), 756–762.

Lay, D. C. (1997), *Linear Algebra and Its Applications*, New York: Addison-Wesley.

Luong, M., Maquin, D., Huynh, C. T., and Ragot, J. (1994), "Observability, Redundancy, Reliability and Integrated Design of Measurement Systems," in *Proceedings of 2nd IFAC Symposium on Intelligent Components and Instruments for Control Applications, SICICA'94*, Budapest, Hungary.

Mili, L., and Coakley, C. W. (1996), "Robust Estimation in Structured Linear Regression," *Annals of Statistics*, 24, 2593–2607.

Mili, L., Phaniraj, V., and Rousseeuw, P. (1990), "High Breakdown Point Estimation in Electric Power Systems," in *Proceedings of the 1990 IEEE International Symposium on Circuits and Systems*, Vol. 3, New Orleans, LA: IEEE, pp. 1843–1846.

Oxley, J. G. (1992), *Matroid Theory*, New York: Oxford University Press.

Rousseeuw, P. J. (1984), "Least Median of Squares Regression," *Journal of the American Statistical Association*, 79, 871–880.

Stanley, G. M., and Mah, R. S. H. (1981), "Observability and Redundancy in Process Data Estimation," *Chemical Engineering Science*, 36, 259–272.

Staroswiecki, M., Hoblos, G., and Aïtouche, A. (2004), "Sensor Network Design for Fault Tolerant Estimation," *International Journal of Adaptive Control and Signal Processing*, 18, 55–72.

Vardy, A. (1997), "The Intractability of Computing the Minimum Distance of a Code," *IEEE Transactions on Information Theory*, 43, 1757–1766.

Wilcox, R. R. (2005), *Introduction to Robust Estimation and Hypothesis Testing*, San Diego, CA: Academic Press.