The University of Florida Sparse Matrix Collection

TIMOTHY A. DAVIS
University of Florida
and
YIFAN HU
AT&T Labs Research

We describe the University of Florida Sparse Matrix Collection, a large and actively growing set of sparse matrices that arise in real applications. The Collection is widely used by the numerical linear algebra community for the development and performance evaluation of sparse matrix algorithms. It allows for robust and repeatable experiments: robust because performance results with artificially-generated matrices can be misleading, and repeatable because matrices are curated and made publicly available in many formats. Its matrices cover a wide spectrum of domains, include those arising from problems with underlying 2D or 3D geometry (as structural engineering, computational fluid dynamics, model reduction, electromagnetics, semiconductor devices, thermodynamics, materials, acoustics, computer graphics/vision, robotics/kinematics, and other discretizations) and those that typically do not have such geometry (optimization, circuit simulation, economic and financial modeling, theoretical and quantum chemistry, chemical process simulation, mathematics and statistics, power networks, and other networks and graphs). We provide software for accessing and managing the Collection, from MATLAB^{TM}, Mathematica^{TM}, Fortran, and C, as well as an online search capability. Graph visualization of the matrices is provided, and a new multilevel coarsening scheme is proposed to facilitate this task.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*linear systems (direct methods), sparse and very large systems*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis, efficiency*; G.2 [**Discrete Mathematics**]: Graph Theory

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: graph drawing, performance evaluation, multilevel algorithms, sparse matrices

1. INTRODUCTION

Although James Wilkinson's foundational work in numerical analysis touched only lightly upon sparse matrix computations, his definition of a sparse matrix is widely used ([Gilbert et al. 1992], for example). Wilkinson defined a matrix as "sparse" if

1st author's address: CISE Dept., Univ. of Florida, Gainesville, FL, USA. davis@cise.ufl.edu. http://www.cise.ufl.edu/∼davis. Portions of this work were supported by the National Science Foundation. 2nd author's address: AT&T Labs Research, Florham Park, NJ, USA. yifanhu@research.att.com. http://www2.research.att.com/∼yifanhu/.

it has enough zeros that it pays to take advantage of them. He actually stated his definition in the negation [Wilkinson 1971]:

> *The matrix may be* sparse*, either with the non-zero elements concentrated on a narrow band centered on the diagonal or alternatively they may be distributed in a less systematic manner. We shall refer to a matrix as* dense *if the percentage of zero elements or its distribution is such as to make it uneconomic to take advantage of their presence.*

In other words, if you can save time or memory (usually both) by exploiting the zeros, then the matrix is sparse. The definition is dependent not only on the matrix, but on the algorithm used on the matrix. For example, a matrix may be "sparse" for an iterative method for solving linear systems or for a graph theoretic algorithm, but not for a sparse factorization method.

This article describes the University of Florida Sparse Matrix Collection (online at http://www.cise.ufl.edu/research/sparse/matrices), which contains sparse matrices arising in a wide range of applications (also referred to as the Collection).

Section 2 gives the motivation for collecting sparse matrices from real applications and making them widely available. Section 3 describes the current state of the Collection and the breadth of problems the matrices represent. Section 4 describes the algorithm used, and new techniques developed, for visualizing the matrices. Section 5 describes the three data formats used for storing the matrices, and the kinds of auxiliary information available for each matrix. Section 6 describes the five methods for searching and downloading matrices of interest: a MATLAB$^{\text{TM}}$interface (UFget), a Mathematica$^{\text{TM}}$interface (ExampleData), a Java interface (UFgui), the matrix web pages (via a standard browser and a web-based search tool), and Amazon Web Services$^{\text{TM}}$. Examples of how the Collection can be used for performance evaluation are given in Section 7. The future of the Collection depends critically upon the submission of new matrices, as discussed in Section 8. Finally, Section 9 summarizes the contributions of the Collection and its associated software.

In this paper, a graph or mesh is said to have 2D or 3D geometry if its vertices have a position on an xy or xyz plane that naturally arises from the problem being solved. A sparse matrix is said to have 2D or 3D geometry if its nonzero pattern is the adjacency matrix of such a graph. The notation $|A|$ refers to the number of nonzeros in a matrix.

2. MOTIVATION

The role of sparse matrices from real applications in the development, testing, and performance evaluation of sparse matrix algorithms has long been recognized. The first established collection was started by Duff and Reid (1970 to 1979, Duff and Reid [1979]), and then compiled into the Harwell-Boeing Collection by Duff, Grimes, and Lewis Duff et al. [1989]. This collection provided the starting point of University of Florida Sparse Matrix Collection. Since the start of our collection in 1991, additional matrices have been added over the years, and other collections have been made, many of which have also been incorporated into the Collection (such as [Bai et al. 1996; 2008; Batagelj and Mrvar 2008; Boisvert et al. 2008; Boisvert et al. 1997; Dumas 2008; Gay 2008; Gould et al. 2008; Mészáros 2008; Mittelmann 2008;

Resende et al. 1995; Rudnyi 2008; Rudnyi et al. 2006; Saad 2008; Schenk 2008]). A widely-available collection such as ours is essential for repeatable experiments.

The Matrix Market [Boisvert et al. 1997] is most similar to the Collection. Both collections include a search tool, and both categorize the matrices by application domain and problem source. Both provide matrices in similar file formats. Both provide a web page for each matrix, with basic statistics and figures. They differ in size, with the Collection containing much larger matrices and 4.5 times as many matrices. The latest matrix added to the Matrix Market was in 2000, whereas the Collection is constantly being updated with new matrices. The largest matrix in the Matrix Market has dimension 90,449 with 2.5 million nonzeros, whereas the largest matrix in the Collection has a dimension of 28 million with 760 million nonzeros. Nearly every matrix in the Matrix Market is also included in the Collection. However, the Matrix Market does include matrix generators; the Collection has no matrix generators.

Nearly all research articles that include a section on the performance analysis of a sparse matrix algorithm include results on matrices from real applications or parameterized matrices that mimic those that could arise in practice. Since maintaining a large collection of matrices from real applications is not trivial, an alternative is first considered, namely, parameterized and random matrix generators.

2.1 Parameterized and random matrices

Randomized or repeatable parameterized sequences of matrices can be easily constructed via simple matrix generators. Examples include: the L-shaped meshes of [George and Liu 1981], least-squares problems from square 2D finite-element meshes [George et al. 1983], non-Hermitian eigenvalue problem (NEP) matrix generators [Bai et al. 1996; 2008], Higham's Matrix Computational Toolbox [Higham 2002] (part of which appears as the `gallery` function in MATLAB), matrices with purely random nonzero patterns (`sprand` in MATLAB [Gilbert et al. 1992]), the YM11 subroutine of [Duff 2001], partially randomized matrices with specific structure (CONTEST toolbox [Taylor and Higham 2009]), and the RMAT generators of [Leskovec et al. 2005] that use Kronecker products to create networks with specified power-law degree distributions.

The prime advantage of random and parameterized matrices is that they are very easy to generate in any size desired. Matrices from real applications are very difficult to generate and it is hard to vary them in size. Another key advantage of random and parameterized matrices is that asymptotic results can sometimes be derived. For example, the nested dissection ordering applied to a 2D s-by-s mesh leads to an asymptotically optimal ordering for sparse Cholesky factorization, with $31(n \log_2 n)/8 + O(n)$ nonzeros in the Cholesky factor L, and requiring $829(n^{3/2})/84 + O(n \log n)$ floating point operations to compute, where $n = s^2$ is the dimension of the matrix [George and Liu 1981].

Purely random matrices may be useful for testing some sparse matrix applications. However, under Wilkinson's definition they are not sparse when factorized by direct methods. With modest assumptions, purely random n-by-n matrices with $O(n)$ nonzero entries require $O(n^3)$ time and $O(n^2)$ memory to factorize, because of catastrophic fill-in [Duff 1974]. Thus, performance obtained on purely random matrices will not indicate how well a sparse matrix factorization method will work

on matrices from real applications. Purely random networks also do not accurately model the characteristics of real networks [Watts and Strogatz 1998].

2.2 Collecting sparse matrices matrices from real applications

While useful, random and parameterized matrices have their limitations. This motivates the development of the Collection, which focuses on matrices from real applications.

The ideal matrix collection would be informally representative of matrices that arise in practice. It should cast a broad net so as to capture matrices from every application domain that relies on sparse matrix methods. For example, matrices arising in circuit simulation (and other network-related domains) differ greatly from matrices arising from the discretization of 2D and 3D physical domains (this will be shown in Section 4.4). Computational fluid dynamics matrices differ from structural engineering matrices, and both are vastly different from matrices arising in linear programming or financial portfolio optimization. The collection should be kept up to date, since matrices of interest grow in size each year as computer memories get larger. New application domains also appear, such as eigenvalue problems arising in web connectivity matrices [Kamvar 2008; Kamvar et al. 2004; Page et al. 1998], which have existed only since the mid 1990's.

Sparse matrix algorithm developers use the matrices in the Collection to develop their methods, since theoretical asymptotic time/memory complexity analysis only goes so far. If there are no matrices available to developers from a given application domain, it is quite possible that when their methods are used in that domain, the performance results will be disappointing. This provides a strong motivation for computational scientists to submit their matrices to a widely available collection such as this one, so that gaps can be avoided. Thus, new application areas are always being added to the Collection.

Our strategy for adding matrices to the Collection is simple, although admittedly ad hoc. The first author maintains a collection of codes for the direct solution of sparse linear systems. End-users of this software are uniformly requested to submit matrices to the Collection. Additional matrices are requested when they are found cited in articles and conference presentations, which includes a wider range of matrices (such as graphs arising in network analysis). Any matrices received are included, unless they are clear duplications of matrices already in the Collection.

This strategy does introduce an unavoidable source of bias, but we have attempted to avoid this bias by relying on matrices collected by others. The Collection includes many sets of matrices collected in this way, such as those collected by Saad, who develops iterative methods for sparse linear systems Saad [2003]. Not all matrices in the Collection arise from a sparse linear system. For example, many linear programming problems have been included in the Collection [Gay 2008; Mészáros 2008; Mittelmann 2008; Resende et al. 1995]. Network and combinatorial problems are also included, such as a matrix of prime numbers from Problem 7 of Trefethen's 100-digit challenge [Trefethen 2002].

Once a matrix is collected, its maintenance is the responsibility of the first author. This guards against arbitrary modifications in the matrices, which can occur when the matrix submitters modify their matrix problems. Repeatability of experiments based on the Collection is thus ensured.

3. DESCRIPTION OF THE COLLECTION

As of March 2010 the University of Florida Sparse Matrix Collection consists of 2272 *problems*, each of which contains at least one sparse matrix (typically just one). It often represents a sparse linear system of the form $Ax = b$, where b is sometimes provided as well as the sparse matrix A. Many other problems are eigenvalue problems, and many matrices come from unstructured graphs and networks with no associated linear system. In some cases, a problem consists of a sequence of closely related matrices, such as Jacobian matrices arising in the solution of a nonlinear system. Some problems include two sparse matrices, such as a stiffness matrix and a mass matrix from a structural engineering eigenvalue problem. In this case, A is the stiffness matrix, and the mass matrix appears as an auxiliary matrix in the problem.

3.1 Matrix groups

The Collection is divided into 157 different matrix *groups*, with more groups added as new matrices are submitted to the Collection. A complete list of these groups is too long to include here; details are given on the collection's web site. Each group typically consists of a set of matrices from a single source. A few of the matrix groups in the Collection consist of entire sets of matrices from another sparse matrix collection. In this case, the group may consist of matrices from very different application areas. For example, the Harwell-Boeing Collection forms a single group [Duff and Reid 1979; Duff et al. 1989].

The group of a problem forms part of the full name of the problem. For example, the full name of the `west0479` problem in the Harwell-Boeing Collection (the `HB` group in the Collection) is `HB/west0479`.

Sources (papers and web sites) for some of the matrix groups are listed in Table I, which gives a flavor of the range of problems the collection contains.

3.2 Matrix authors and editors

The 2272 matrices in the Collection come from 359 different authors and 50 different editors. A matrix *editor* is the one who collected the matrix for inclusion into any established collection. A matrix *author* is the one who created the matrix. The editor and author are included in the meta-data for each problem.

3.3 Matrix `kind`

The matrix group classifies a matrix by its source, but many groups include matrices from different application areas. Also, matrices in a single application area can be present in different groups.

To cross-index the matrices in the Collection, all problems are tagged with a string, `kind`, which indicates the application domain of the problem. The group and `kind` of a problem are not the same. For example, the `HB/west0479` problem is in the `HB` group, and its `kind` is tagged as a "chemical process simulation problem." Five other groups include chemical process simulation problems, namely, the `Bai`, `Grund`, `Mallya`, `VanVelzen`, and `Zitney` groups.

A complete list of the `kind` strings for all problems is given in Table II. The table is split into two categories (problems with no underlying geometry and problems

Table I. Partial list of sources of matrices

Group	Description	Citation
Bai	Non-Hermitian Eigenvalue Problems	[Bai et al. 1996]
Pajek	Pajek Networks	[Batagelj and Mrvar 2008]
Mulvey	Multistage stochastic financial modeling	[Berger et al. 1995]
(various)	The Matrix Market (collection)	[Boisvert et al. 1997]
Bomhof	Univ. of Utrecht circuit simulation	[Bomhof and van der Vorst 2000]
Bydder	MRI reconstruction	M. Bydder, UCSD
HB	Harwell-Boeing Collection	[Duff et al. 1989]
JGD_*	Combinatorial problems	[Dumas 2008]
ATandT	Frequency domain, nonlinear circuits	[Feldmann et al. 1996]
LPnetlib	NETLIB Linear Programming Problems	[Gay 2008]
GHS_*	Symmetric sparse matrix benchmarks	[Gould et al. 2008]
Gupta	Linear programming problems	[Gupta 1996]
Kamvar	Stanford/Berkeley Web Matrices	[Kamvar et al. 2004]
Sandia	Xyce circuit simulation matrices	[Keiter et al. 2003]
Kemelmacher	Computer vision problems	[Kemelmacher 2005]
Mesazros	Linear programming test set	[Mészáros 2008]
Wang	2D and 3D semiconductor physics	[Miller and Wang 1991]
Mittelmann	Linear programming test set	[Mittelmann 2008]
Qaplib	QAPLIB, quadratic assignment	[Resende et al. 1995]
Oberwolfach	Oberwolfach Model Reduction	[Rudnyi et al. 2006]
(various)	SPARSKIT matrix collection	[Saad 2008]
Schenk_*	Univ. of Basel Collection	[Schenk 2008]
vanHeukelum	DNA electrophoresis matrices	[van Heukelum et al. 2002]
Zitney	Chemical engineering problems	[Zitney 1992]
Mallya	Chemical engineering problems	[Zitney et al. 1996]

with 2D or 3D geometry) and sorted by the number of matrices in each category.

The Collection contains 68 matrices with random nonzero pattern. They appear in the Collection only because they already occur as widely-used test problems in another collection that was subsequently added to the Collection.

3.4 Matrix statistics

Figures 1 and 2 plot the matrix size (dimension and number of nonzeros) versus the year in which the matrices were created. The solid line in the figures is the cumulative sum of the data plotted. Both figures show an exponential growth in problem sizes, similar to how computer memory sizes have grown since 1970.

The outlier matrix in 1971 is from the Edinburgh Associative Thesaurus, located at http://www.eat.rl.ac.uk and obtained from the Pajek data set [Batagelj and Mrvar 2008]. It is a graph with 23,219 vertices and 325,592 edges that was first constructed in 1971 [Kiss et al. 1973].

Figure 3 plots two histograms of the overall distribution of matrices in the Collection.

4. VISUALIZING THE COLLECTION

Many basic facts, including symmetry, structural rank, ordering statistics, as well as a plot of the sparsity pattern, are given for each matrix in the collection. However these facts alone do not always give sufficient information about the matrix. For example, does this matrix come from an application involving 2D or 3D mesh?

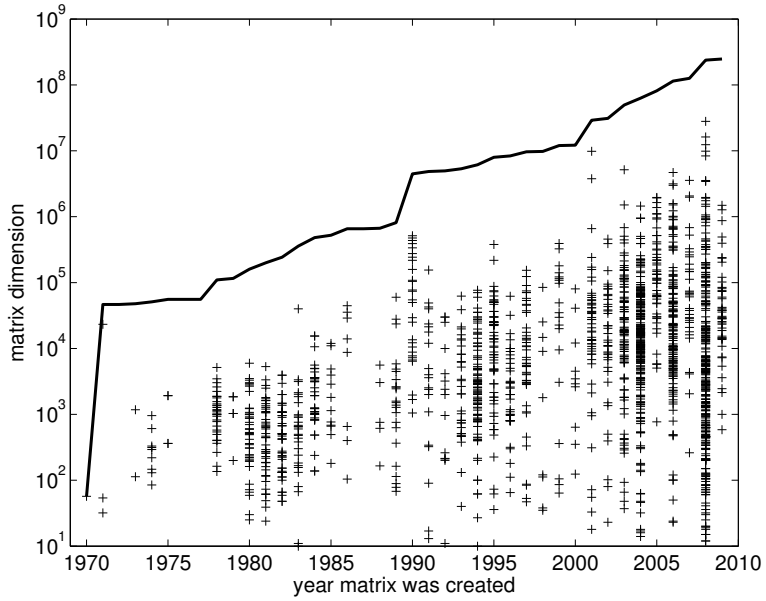


Fig. 1. Matrix dimension (the largest of row/column dimension if rectangular) versus year created. The solid line is the cumulative sum.

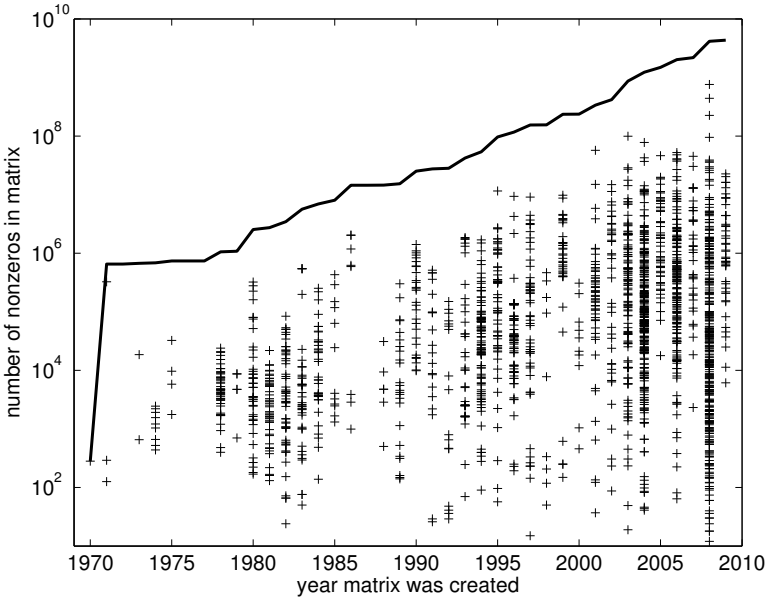


Fig. 2. Number of nonzeros in each matrix versus year created. The solid line is the cumulative sum.

Table II. Summary of `Problem.kind` for all 2272 problems

1516	**problems with no 2D/3D geometry**
342	linear programming problem
299	combinatorial problem
251	circuit simulation problem
135	optimization problem
88	directed graph
70	chemical process simulation problem
68	economic problem
68	random graph
61	theoretical/quantum chemistry problem
56	power network problem
23	least squares problem
23	undirected graph
11	counter-example problem
10	statistical/mathematical problem
8	bipartite graph
4	frequency-domain circuit simulation problem
756	**problems with 2D/3D geometry**
288	structural problem
166	computational fluid dynamics problem
94	2D/3D problem (other than those listed elsewhere)
44	electromagnetics problem
42	model reduction problem
35	semiconductor device problem
31	thermal problem
28	materials problem
13	acoustics problem
12	computer graphics/vision problem
3	robotics problem

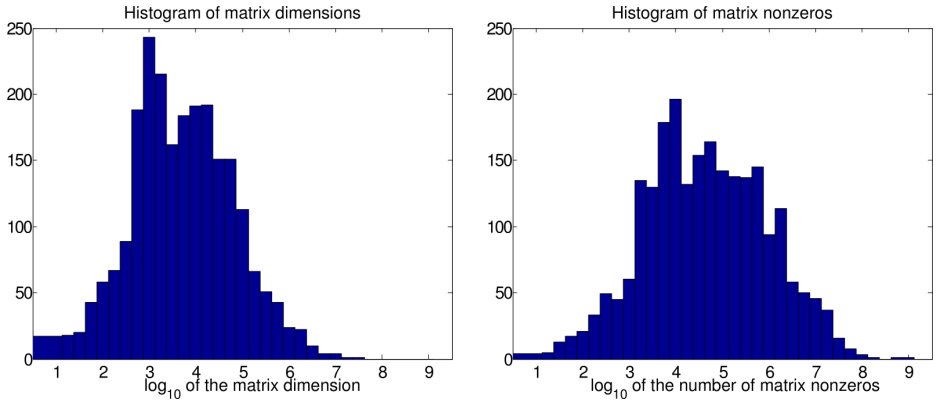


Fig. 3. Overall histograms of matrix dimensions and nonzeros

Or from one involving a small-world network? Are there other structures in the underlying applications that are not discernible from a plot of the sparsity pattern, bearing in mind that a matrix plot depends on the ordering of the matrix?

To help answer these questions, a visualization of each matrix in the form of graph drawing is provided. If the matrix is structurally symmetric, it is taken as the adjacency matrix of an undirected graph, where two vertices i and j are connected if the (i, j)-th entry of the matrix is nonzero. Rectangular or structurally unsymmetric matrices are treated as bipartite graphs. More specifically, the augmented matrix

$$\begin{pmatrix} 0 & A \\ A^T & 0 \end{pmatrix}$$

is used as the adjacency matrix of an undirected bipartite graph whose vertices are composed of the rows and columns of the matrix. We provide two graphs for square matrices with unsymmetric structure: the graph of $A + A^T$ and the augmented matrix above. The layout produced by our graph drawing algorithm only takes account of the nonzero pattern, not the values.

The basic algorithm used for drawing the graphs is a multilevel force-directed algorithm [Hu 2005]. However this algorithm fails to produce informative drawings for some matrices in the Collection (as we demonstrate in Section 4.2). We propose in Section 4.3 a new coarsening scheme to deal with these cases. In the following we briefly describe the basic algorithm, followed by the new coarsening scheme.

4.1 The graph drawing algorithm

The basic algorithm [Hu 2005] employs a force-directed model [Fruchterman and Reingold 1991]. Vertices are treated as particles with electrical charges that push each other away. This repulsive force $F_r(i, j)$ between any two vertices i and j is proportional to the inverse of their distance,

$$F_r(i, j) = \frac{K^2}{\|x_i - x_j\|}, \ i \neq j.$$

Here K is a parameter representing the nominal ideal distance of an edge, and the coordinates of vertex i are x_i. At the same time, vertices that share an edge are attracted to each other by a spring force,

$$F_a(i, j) = \frac{\|x_i - x_j\|^2}{K}, \ i \text{ and } j \text{ share an edge},$$

that is proportional to the square of their distance. The minimal energy configuration of this physical system is taken as the optimal drawing of the graph.

An iterative algorithm is used to attempt to find the optimal configuration. Starting from a random layout, for each vertex, the combined repulsive and attractive forces are calculated, and the vertex is moved along the direction of the force, with the distance moved controlled by a step length. This process is repeated, with the step length gradually reduced, until the positions stabilize. This simple force-directed process works well for small graphs. However, for large graphs, the process is likely to give suboptimal layout, due to the fact that this physical system of springs and electrical charges has many local minimum configurations. Therefore

we employ a multilevel scheme to provide a "global view" and to help find a global optimum.

The multilevel approach has three distinctive phases: coarsening, coarsest graph layout, and finally, prolongation and refinement. In the coarsening phase, a series of coarser and coarser graphs, $\mathcal{G}^0, \mathcal{G}^1, \ldots, \mathcal{G}^l$, are generated, the aim is for each coarser graph $\mathcal{G}^{k+1}$ to encapsulate the information needed to layout its "parent" $\mathcal{G}^k$, while containing roughly a fraction t, or less, of vertices (we set $t = 0.75$). The coarsening continues until a graph with only a small number of vertices is reached. The optimal layout for the coarsest graph can be found cheaply. The layout on the coarser graphs are recursively interpolated to the finer graphs, with further refinement at each level.

Another important ingredient to allow very large graphs to be drawn is the Barnes-Hut force approximation scheme [Barnes and Hut 1986]. In this scheme, a nested space partitioning data structure is used to approximate the all-to-all repulsive force so as to reduce the quadratic complexity to log-linear, under certain assumptions.

4.2 A failure in popular coarsening schemes

The aforementioned multilevel algorithm was found to work well for many test graphs from the graph drawing literature [Hu 2005]. However, when applied to the Collection, we found that for some matrices, the multilevel process broke down by terminating too early when the graph was still large, because the coarsening procedure could not produce a coarse graph that has sufficiently less vertices. Consequently the resulting drawing was poor. An example is shown in Figure 4. To the left of the figure is the sparsity pattern of the `Gupta/gupta1` matrix. From this plot we can conclude that this matrix describes a graph of three groups of vertices: those represented by the top 1/3 of the rows in the plot, the middle 1/3 of the rows, and the rest. Vertices in each group are all connected to a selected few in that group, these links are seen as dense horizontal and vertical bars in the matrix plot. At the same time, vertices in the top group are connected to those in middle group, which in turn are connected to those in the bottom group, as represented by the off-diagonal lines parallel to the diagonal. However, the graph drawing to the right of Figure 4 shows none of these structures. The three different groups of vertices are co-mingled together.

4.3 A new coarsening scheme

Analyzing the multilevel process (Table III), we found that while up to level 2, a significant reduction in the number of vertices were achieved as the graphs were coarsened (a reduction from 31802 vertices to 12034 vertices), between level 2 and level 3, the number of vertices hardly changed ($11088/12034 = 0.92 > t$). Consequently we had to stop the coarsening process and use the iterative force-directed algorithm on the large coarsest level graph with 11088 vertices. As expected, the result in Figure 4 was very poor.

This matrix exemplifies many of the problematic matrices: they contain star-graph like substructures, with many vertices all connected to a few vertices. Such structures pose a challenge to the popular graph coarsening schemes, in that these schemes are not able to coarsen them adequately. One of these schemes is based

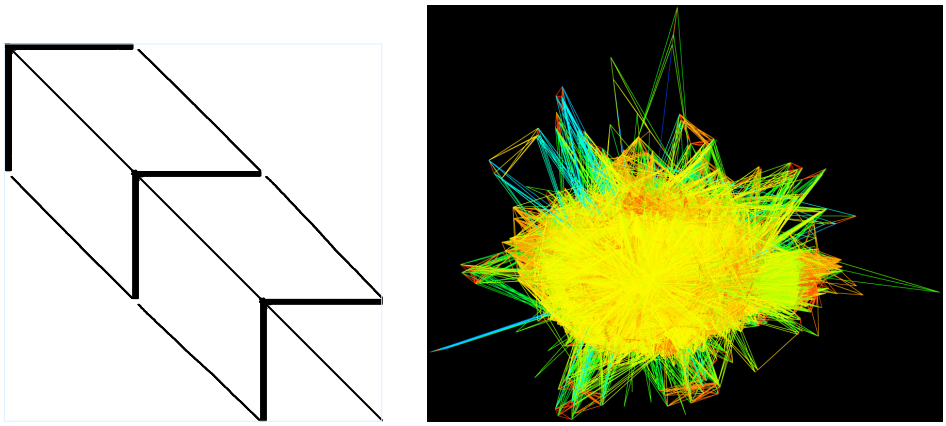


Fig. 4. Matrix plot of `Gupta/gupta1` matrix (left) and the initial graph drawing (right). The relative length of an edge (as compared to the average edge length) determines its color. Relatively long edges have cool colors and short edges have warmer colors, from blue (cool) to green, yellow, then red.

Table III. The process of coarsening on `Gupta/gupta1` matrix.

level	$\|V\|$	$\|E\|$
0	31802	2132408
1	20861	2076634
2	12034	1983352
3	11088	stopped

on maximal independent edge set (MIES). An MIES is the largest set of edges that does not share a vertex. An MIES-based coarsening scheme works by finding an MIES, then merging the end vertex pairs of the edges in the set. This gives us a graph with fewer vertices and edges. Unfortunately for star-graph like structure, this scheme does not coarsen sufficiently. For example, Figure 5 (left) shows such a graph, with $k = 10$ vertices on the out-skirts all connected to two vertices at the center. Because of this structure, any MIES can only contain two edges (the two thick edges in Figure 5). When the end vertices of the MIES are merged to the center of each edge, the resulting graph has only two fewer vertices. Therefore if k is large enough, the coarsening can be arbitrarily slow ($k/(k+2) \to 1$ as $k \to \infty$). Other popular coarsening schemes, such as the one based on maximal independent vertex set, would suffer from the same slow coarsening problem.

Having found the root cause of the problem, we propose a new coarsening scheme in the following. The scheme works by finding vertices that share the same neighbors (these vertices are known as supervariables in the numerical analysis literature [Duff and Reid 1996], or as modules in the graph theory literature [McConnell and Spinrad 1999]), then matching pairs of these vertices. A usual MIES scheme then matches the remaining unmatched vertices. Finally the matched vertices are merged to arrive at the coarsened graph. The scheme is able to overcome the slow coarsening problem associated with graphs having star-graph like substructures.

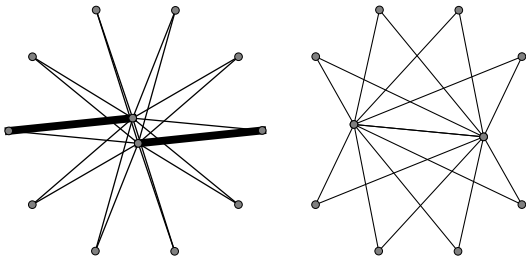


Fig. 5. A maximal independent edge set based coarsening scheme fails to coarsen sufficiently a star-graph like structure. Left, a maximal independent edge set (thick edges); right: when merging the end vertices of the edge set at the middle of these edges, the resulting coarsened graph only has 2 less vertices.

Applying this scheme to the graph in Figure 6 (left) resulted in a graph with 1/2 the number of vertices (Figure 6 right). With this new coarsening scheme, we are able to layout many more graphs aesthetically. For example, when applied to the `Gupta/gupta1` matrix, the drawing at Figure 7 reveals the correct visual structures as we expected, with three groups of vertices (left, middle, right), each connected to a few within the group, and a linear connectivity relation among the groups (left group connects to the middle, which in turn connects to the right).

It is worth noting that coarsening schemes such as MIES have been frequently used in the graph drawing literature (e.g., [Hachul and Jünger 2004; Hu 2005; Walshaw 2003]). Extensive testing on the Collection revealed their limitation and allowed us to propose the improved scheme. This is an example that illustrates the value of such a large collection.

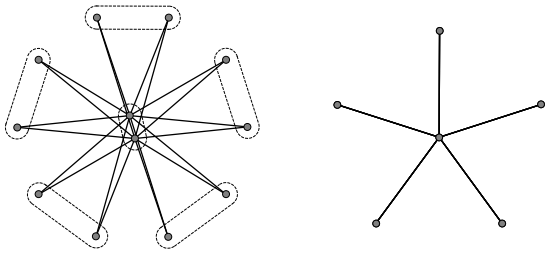


Fig. 6. Matching and merging vertices with the same neighborhood structure (left, with dashed line enclosing matching vertex pairs) resulted in a new graph (right) with 1/2 the number of vertices.

4.4 Additional example graph drawings

Two additional examples of our force-directed graph drawing are given in Figures 8-10. The nonzero pattern of a 3D model of a Boeing/Sikorsky RAH-66 Comanche helicopter in Figure 8 does not provide any visual cues as to what this matrix represents. However, the structure is evident in the two graph drawings in Figure 9, which compares the actual 3D geometry (obtained from Alex Pothen) with our

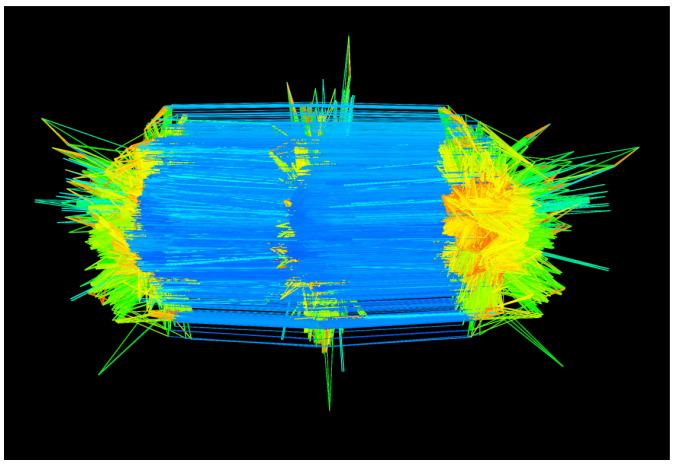


Fig. 7. After applying the new coarsening scheme, the force-directed graph drawing correctly reflects the structure of the matrix, with three distinct groups of vertices in the middle, and to the left and right; two sparse sets of long edges linking these three groups.

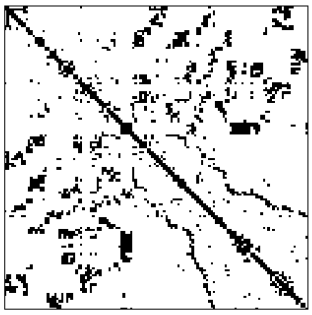


Fig. 8. Sparse matrix pattern of a RAH-66 Comanche helicopter

force-directed graph. The latter clearly shows that this problem comes from the discretization of a 2D/3D object, and it bears some resemblance to the actual object. Figure 10 is the nonzero pattern of matrix arising from circuit simulation (left) and its force-directed graph drawing (right). The graph drawing is typical of matrices with a network topology that has no 2D/3D geometry.

In addition to providing valuable information for users of the Collection, visualization of the matrices reveals their intrinsic beauty and has attracted the attention of the media [IEEE Spectrum editorial staff 2010; Labarre 2010; Gorter and Kasky 2010], instances where the beauty and art of mathematics are appreciated by the general public.

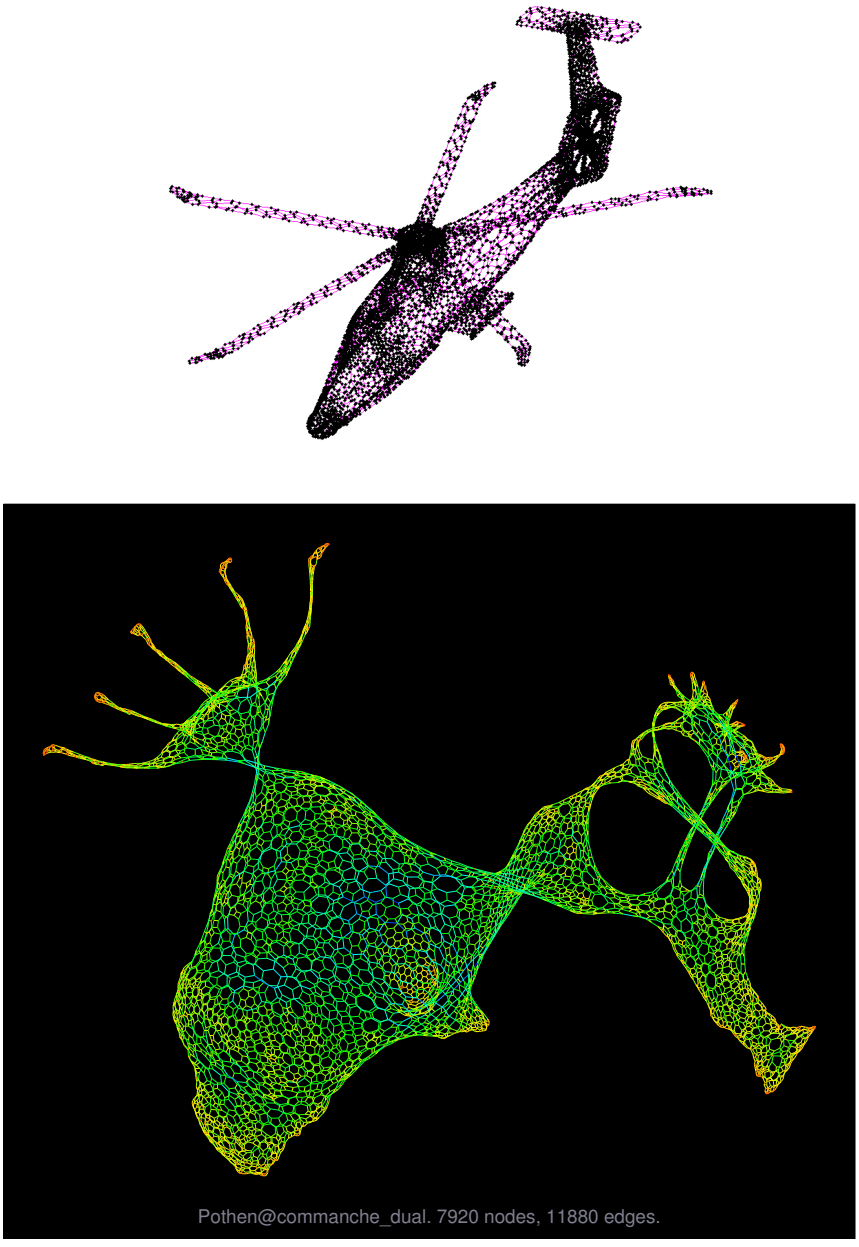


Fig. 9. Graph of a RAH-66 Comanche helicopter, using given 3D coordinates (top) and its force-directed graph drawing (bottom). The force-directed graph drawing reveals a fine fish-net-like mesh. In the force-directed graph, the helicopter's 5 rotors are on the top left, the tail rotor is to the right, and the nose is to the bottom left. Most of the edges are the same length and the fine structure of each local neighborhood of vertices is very regular. These features are typical of matrices representing 2D/3D problems.

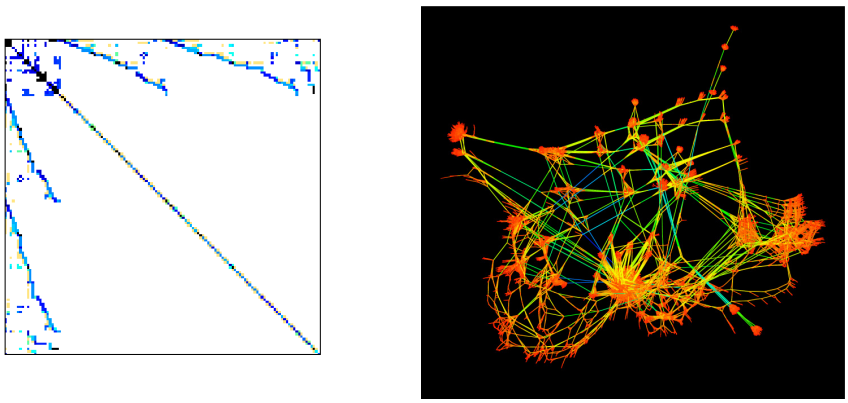


Fig. 10. Sparse matrix pattern of an electronic circuit from Steve Hamm, Motorola (left) and its force-directed graph (right). The graph has many short edges (shown in red) and some very long edges (shown in green and blue), which is typical of matrices that represent a general network rather than a problem with 2D/3D geometry.

5. MATRIX DATA FORMATS

The matrix problems appear in three formats: MATLAB MAT-file, Matrix Market (MM) [Boisvert et al. 1997], and Rutherford-Boeing (RB) [Duff et al. 1997]. The matrices and all auxiliary data are identical in all three formats, down to the very last bit (note that the MATLAB *.mat format is in binary). The following data appears each matrix problem:

—`name`: the matrix name (`HB/west0479`, for example).
—`title`: a short descriptive title.
—`A`: the sparse matrix itself.
—`id`: a unique integer identifier permanently assigned to the matrix.
—`date`: the date the matrix was created.
—`author`: the matrix author (left blank if unknown).
—`ed`: the matrix editor.
—`kind`: the matrix *kind*-string (Table II).

The following optional fields are present in some problems:

—`Zeros`: When a matrix is submitted we are sometimes given explicit entries that are numerically zero. In the MM and RB formats, these entries are held in the matrix itself. Since MATLAB drops explicit zeros, the `Zeros` field in the MATLAB format is a binary matrix where the ijth entry is 1 if the submitter provided the explicit entry $a_{ij} = 0$.
—`b`: a right-hand side (sparse or dense).
—`x`: the solution to $Ax = b$ (sparse or dense).
—`notes`: additional text describing the problem.
—`aux`: any additional data. The contents of this are problem-dependent. Examples include the l, h, and c vectors for a linear-programming problem (minimize $c^T x$

subject to $Ax = b$ and $l \leq x \leq h$), an n-by-3 array of xyz coordinates of the vertices, an array of n strings with the names of each vertices, a sparse mass matrix for a generalized eigenvalue problem, etc.

In the MAT-format, the entire problem is held as a MATLAB `struct`, and stored in its own MAT-file.

The MM format [Boisvert et al. 1997] stores a sparse matrix as a collection of triplets (i, j, a_{ij}), with the row index, column index, and numerical value. In the MM format, triplets appear one per line in any order and duplicates are permitted. However, our MM files are all sorted, with no duplicate entries. The MM format permits comments in the matrix file, and most of the additional problem characteristics (`name`, `id`, `notes`, etc.) appear as structured comments.

The RB format [Duff et al. 1997] stores a sparse matrix in compressed-column form, as three vectors `Ap`, `Ai`, and `Ax`. The row indices and numerical values of entries in the jth column are located in positions `Ap(j)` through `Ap(j+1)-1` of `Ai` and `Ax`, respectively. This format does not allow for structured comments, so this information is held in a separate file. The RB format can also represent a finite-element matrix in unassembled form, but very few matrices are ever submitted in this form, and they are all small. Thus, for simplicity, the Collection does not use this format. For the MM and RB formats, the matrix file and any auxiliary matrices are all kept in a single directory, then archived as a single ZIP file.

The SuiteSparse package (http://www.cise.ufl.edu/research/sparse) includes software in MATLAB, C, and Fortran, for reading and writing the matrices and for creating and maintaining the Collection.

6. ACCESSING THE COLLECTION

There are five methods for accessing matrices in the Collection on the web.

6.1 UFgui: Java interface

UFgui is a Java-based application for browsing, selecting, and downloading matrices from the primary web site for the Collection. A screenshot is shown in Figure 11. It provides a selection mechanism for choosing matrices with specific features (size, symmetry, number of nonzeros, whether it has 2D/3D geometry or not, matrix group, and matrix kind). The selected matrices act as a set of id's, where clicking select acts as a set union, and deselect acts as a set difference. For example, to select all real square matrices except for those arising in circuit simulation, select real and square, and then click select. Then select chemical process simulation and click deselect. Clicking download then downloads all selected matrices that have not yet been downloaded. Right-clicking the table of matrices provides options for exporting the list of selected matrix id's as a spreadsheet or a MATLAB M-file. As matrices are selected in the table, an image of the nonzero pattern of the most recently-selected matrix is displayed. The table also indicates which matrices have already been downloaded, under the mat, MM, and RB column headings. In Figure 11, the `HB/will199` matrix has already been downloaded into `/Users/davis/sparse/UFget`, in both MAT and MM formats, but not in RB format.

Clicking on a column heading sorts the table by that column. The sort is stable,

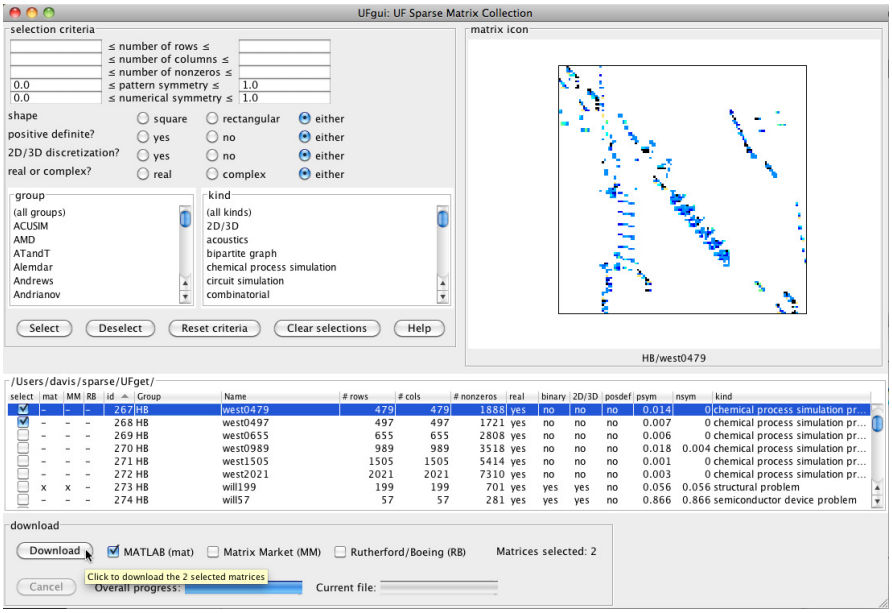


Fig. 11. UFgui: Java interface to the Collection

in the sense that ties are broken according to the previous sort. Thus, to view matrices grouped by their `2D/3D` characteristic, and then sorted by the number of rows, the user can first click to sort on the number of rows, and then click the `2D/3D` column heading to sort by that characteristic.

6.2 UFget: MATLAB interface

UFget is our MATLAB interface to the Collection. `Prob=UFget('HB/west0479')` loads the matrix of that name into the MATLAB workspace, downloading it if necessary. `Prob=UFget(267)` does the same, using the problem id instead. The index of the entire collection is loaded with `index=UFget`, which automatically updates itself every 90 days so that new matrices can be included. This index contains statistics about each matrix and can be searched via simple MATLAB expressions. For example, the following MATLAB code downloads all square matrices with full structural rank whose nonzero pattern is at least 50% symmetric, in increasing order of number of nonzeros. It then computes a fill-reducing ordering with AMD [Amestoy et al. 1996; 2004] on the pattern of $A + A^T$ (ignoring numerical cancellation), and determines the number of nonzeros in the Cholesky factorization.

```
index = UFget ;
matrices = find (index.nrows == index.ncols & ...
    index.sprank == index.nrows & ...
    index.pattern_symmetry > 0.5) ;
[ignore i] = sort (index.nnz (matrices)) ;
matrices = matrices (i) ;
nmat = length (matrices) ;
for k = 1:nmat
```

```
    Problem = UFget (matrices (k)) ;
    A = spones (Problem.A) + spones (Problem.A') ;
    p = amd (A) ;
    anz = nnz (A) ;
    lnz = sum (symbfact (A (p,p))) ;
end
```

6.3 Mathematica interface

Mathematica has built-in support to the Collection via its `ExampleData` function. This is a general function that allows Mathematica users to download data of many different kinds (e.g., financial data; country data; chemistry data), with sparse matrices from the Collection one of these.

For example, `M = ExampleData[{"Matrix","HB/west0479"}]` assigns the matrix of that name to the variable `M`, downloading it through the Internet from a mirror server at Wolfram Research Inc., if necessary. Properties of the matrix can be queried via `ExampleData[{"Matrix", "HB/west0479"}, "Properties"]`, without actually downloading the matrix itself. `ExampleData["Matrix"]` lists all the matrices in the Collection.

6.4 Via a web browser

The primary web site is http://www.cise.ufl.edu/research/sparse/matrices, with a link to an online form shown in Figure 12 that allows users to search by keywords, as well as by matrix type, structure, dimensions or other statistics. For example, entering "circuit" as a search term provides a list of all circuit-related matrices, with basic statistics, a thumbnail graph, a link to the web page of each matrix, as well as links for downloading the matrices in the three supported formats.

Each matrix has its own web page, with matrix statistics, any notes about the matrix, download links, an image of the nonzero pattern of the matrix, and a drawing of the graph using the force-directed graph drawing method discussed in Section 4. The main index page includes a sample of 49 thumbnail-sized snapshots of these graphs, shown in Figure 13, which gives a visual indication of the diversity and complexity of matrices in the Collection. Each matrix group has a web page, with an index showing thumbnail matrix images and graphs, and basic statistics.

6.5 Amazon Web Services

The entire Collection is mirrored by Amazon Web Services™as a Public Data Set at http://aws.amazon.com. The data sets provided by Amazon can be seamlessly integrated into cloud-based applications. An application can be built that accesses the matrices directly without having to explicitly download them.

7. EXAMPLE USES OF THE COLLECTION

A collection such as the one described here can answer many questions about the design, analysis, and performance of sparse matrix algorithms that cannot be answered by theoretical analyses or artificially-generated matrices. Matrices obtained from (or available in) this collection have been used in a wide range of published experimental results. For a small sample, see [Amestoy et al. 2007; Amestoy and Puglisi 2002; Benzi and Tuma 1998; Brainman and Toledo 2002; Demmel et al.

Search The University of Florida Sparse Matrix Collection

Fill in one or more of the following fields and click on "search"

circuit Keyword (contained in group, name, kind, author, editor and notes)
More than [ ] but less than [ ] rows.
More than [ ] but less than [ ] columns.
More than [ ] but less than [ ] nonzeros.
More than [ ] but less than [ ] % nonzero pattern symmetry.
Structure Any
Positive definite? Any
Type Any
Group: [ ] Name: [ ]
Date: after [ ] but before [ ] year
Search

Search results: 262 matrices found.

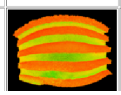
hit	group	name	image	rows	cols	nonzeros	kind	Matrix Market format	Matlab format	Rutherford/Boeing format
1	AMD	G3_circuit		1585478	1585478	7660826	circuit simulation problem	download	download	download

Fig. 12. Online search interface

1999; Duff and Koster 1999; Duff and Pralet 2005; Gupta 2002; Schulze 2001]. Articles that do not use standard benchmarks such as the Collection typically use matrices that could arise in practice, such as the 5-point discrete Laplacian on a regular mesh ([Ruesken 2002], for example).

Three examples of the kinds of questions a set of real matrices can answer are given here: the average-case time complexity of the minimum degree ordering algorithm, the typical fill-in in a Cholesky factorization using the AMD or METIS orderings, and the trade-off between supernodal and non-supernodal sparse Cholesky factorization methods [Chen et al. 2008].

7.1 First example: average-case run time of minimum degree

The *minimum degree ordering algorithm* is a heuristic for finding a permutation P such that the Cholesky factorization of PAP^T is sparser than that of A. The running time of the minimum degree ordering algorithm is notoriously difficult to analyze. Under modest assumptions, a loose worst-case upper bound on the run time of the approximate-minimum-degree (AMD) variant of this method is given by

$$O\left(\sum_{k=1}^{n} |L_{k*}| \cdot |(PAP^T)_{k*}|\right), \tag{1}$$

where P is the permutation found by AMD, L is the Cholesky factor of PAP^T, and $|x|$ denotes the number of nonzeros in a vector or matrix [Amestoy et al. 1996; 2004]. The bound does not consider the speedup obtained by exploiting supernodes, which has not been analyzed because the graph changes unpredictably during the elimination.

A single dense row or column of A leads to a run time of at least $\Omega(n^2)$, but AMD includes a preprocessing step that removes any row or column with degree $10\sqrt{n}$ or larger (the effect of this step is not accounted for in (1)). Still, a host of

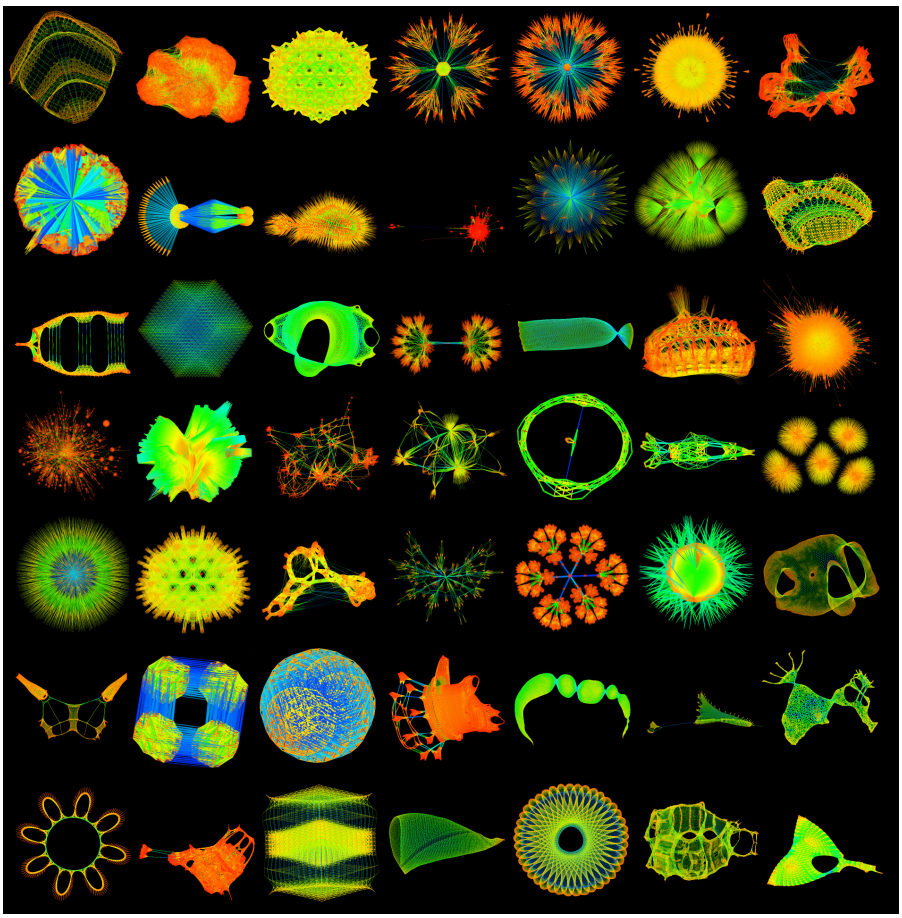


Fig. 13. A sample of matrices from the Collection, for the purpose of illustrating the complexity and diversity of matrices arising in real applications

nearly-dense rows/columns could lead to unacceptable ordering time. Is the bound (1) reached in practice? What is the average-case time complexity of AMD?

Determining the theoretical average-case time complexity is beyond the scope of any analysis that has been done for this algorithm, so the best that can be done is to test the method on a set of "average" sparse matrices that arise in practice. The results of ordering $A + A^T$ (ignoring numerical cancellation in the matrix addition) on all square matrices in the Collection (as of November 2006) whose nonzero pattern is symmetric (or mostly so) are shown in Figures 14 and 15. The code for selecting this set is shown in Section 6.2. Each matrix is a single dot in Figures 14 and 15.

The results are split into two sets: matrices from problems with 2D/3D geometry, and problems without any underlying geometry. A best-fit line is shown (admittedly not a very good fit). Figure 15 indicates that the average-case run time may be slightly super-linear in $|A|$ (at least for 2D/3D problems), with the best-fit line being

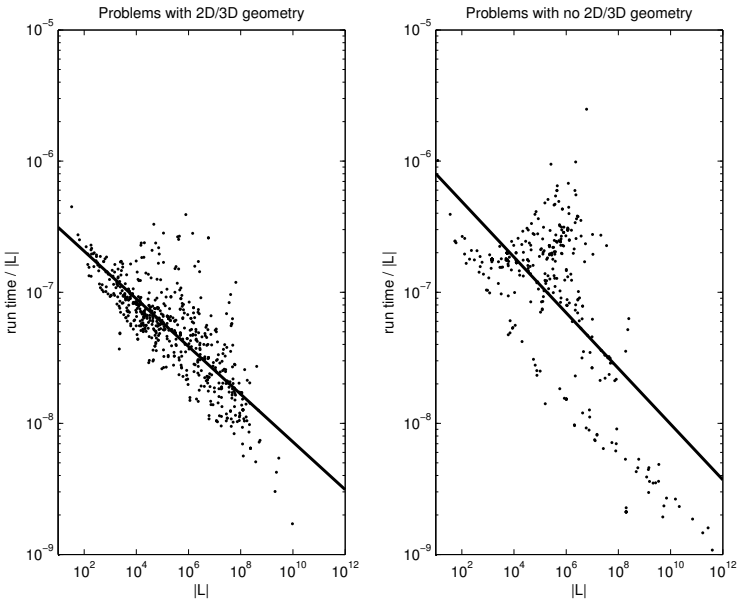


Fig. 14. AMD run time in seconds / $|L|$, as a function of $|L|$

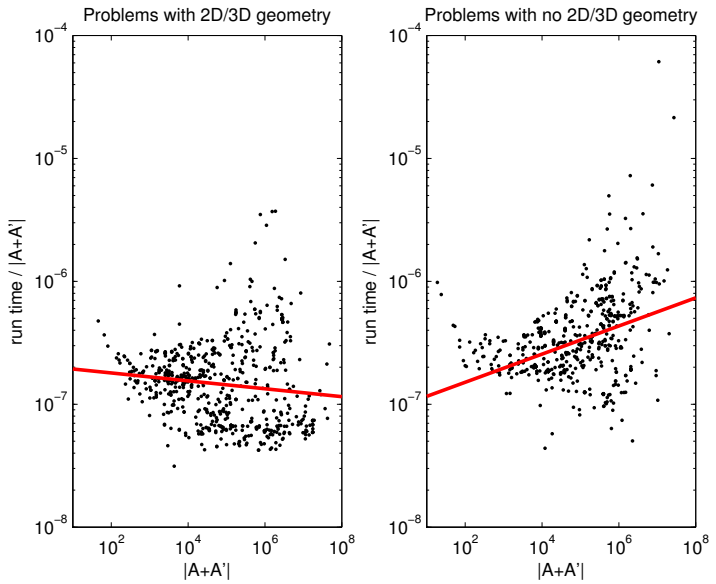


Fig. 15. AMD run time in seconds / $|A + A^T|$, as a function of $|A + A^T|$

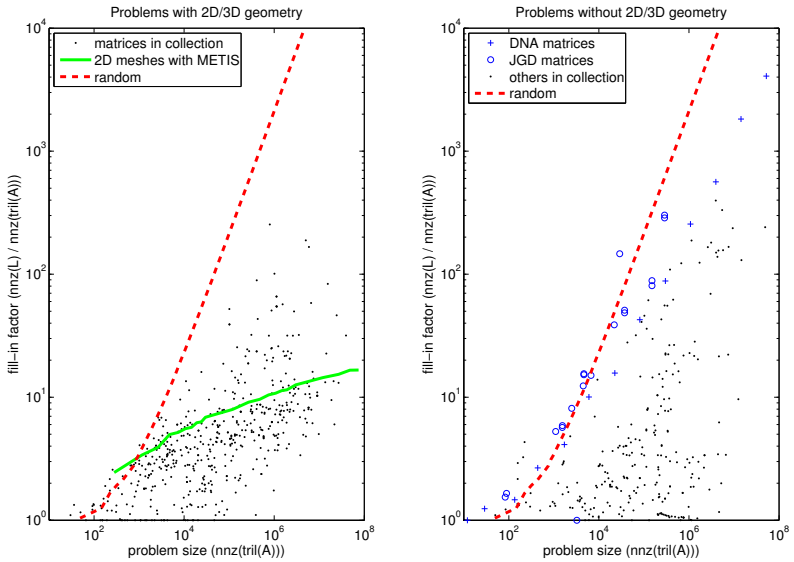


Fig. 16. AMD/METIS fill-in factor, as a function of the number of nonzeros in the lower triangular part of A, nnz(tril(A))

$O(|A|^{1.1})$ for problems with 2D/3D geometry, and $O(|A|^{1.4})$ otherwise. However, the results in the Figure 15 are very noisy so this conclusion cannot be made strongly. Figure 14 gives a clear result. It shows that $O(|L|)$ is a very loose upper bound on the average case run time (excluding a few worst-case examples), even though $O(|L|)$ is already a much lower bound than (1). The best-fit line is $O(|L|^{0.66})$ for the 2D/3D case, and $O(|L|^{0.61})$ for problems with no 2D/3D geometry.

7.2 Second example: typical fill-in

To illustrate the importance of the non-random structure in a matrix, consider Figure 16, which plots the quality of the ordering from AMD or METIS [Karypis and Kumar 1998] (whichever gives the best result). For this figure, the y-axis is the number of nonzeros in L divided by the number of nonzeros in the lower triangular part of A, denoted $nnz(tril(A))$ (the latter is also the x-axis). A metric of 1.0 means that no fill-in occurred. This plot includes all matrices with perfectly symmetric pattern in the collection as of March 2010.

Superimposed on each of the two plots in Figure 16 is a dashed line for matrices with random nonzero pattern from the MATLAB `sprandsym` function, with an average of 5 nonzeros per row or column including a zero-free diagonal. This is the same number of entries as a matrix arising from the discretization of a 2D mesh. The left figure includes a solid line which is the METIS result on a square 2D mesh (recall that for these matrices, $|A| = 5n$ and $|L| = 31(n \log_2 n)/8 + O(n)$). The METIS result on square 2D meshes indicates that most matrices from real applications with 2D/3D geometry have a fill-in factor that is not much different

than these simple square meshes.

On the right plot, two sets of matrices are highlighted that have particularly high fill-in. The +'s are from a DNA electrophoresis problem [van Heukelum et al. 2002], and the o's are from J.-G. Dumas' collection of combinatorial mathematical problems [Dumas 2008]. Both sets of matrices have very irregular patterns, and seem to approach the level of fill-in from matrices with random pattern. The DNA and JGD matrices do not have 2D/3D geometry, so they do not appear in the left plot. Other that these two sets of matrices, the fill-in from matrices from real applications seems to be asymptotically lower than fill-in in random matrices.

The outlier in the left plot in Figure 16 with the highest fill-in factor is the `GHS_indef/sparsine` matrix, a structural optimization matrix [Gould et al. 2008]. It is labeled as a problem with 2D/3D geometry in the Collection when added (by this author), but this label might be incorrect. Optimization problems often have irregular structure and no underlying geometry, although a mechanical structural optimization problem would presumably have a 2D or 3D geometry. The graph drawing of this matrix can be seen in Figure 13, in the top row, second column. It is very irregular and is visually similar to the linear programming problem in the top right corner of the same figure (the `Andrianov/lpl1` matrix). The `sparsine` graph is very different from the regular structure of a finite-element matrix (`Alemdar/Alemdar` in the top left corner, for example), which may explain why it is an outlier in the left plot in Figure 16.

7.3 Third example: BLAS trade-off in sparse Cholesky factorization

CHOLMOD is a sparse Cholesky factorization and update/downdate package that appears in `x=A\b` and `chol(A)` in MATLAB, when `A` is sparse and symmetric positive definite [Chen et al. 2008; Davis and Hager 2009]. It includes two sparse Cholesky factorization methods: a BLAS-based supernodal method [Ng and Peyton 1993] and an up-looking non-BLAS-based method [Davis 2005]. The two methods are included because a BLAS-based method is slower for very sparse matrices (tridiagonal ones, for example). The dense matrix operations in the BLAS gain their performance advantage when the ratio of floating-point work to memory traffic is high. Thus, we predicted that the ratio of the number of floating-point operations over the number of nonzeros in L would be a good way to automatically select the appropriate method. Both of these terms are available from the symbolic analysis, prior to numerical factorization. A similar metric was used to compare the BLAS-based SuperLU method [Demmel et al. 1999] with its non-BLAS based precursor, GPLU [Gilbert and Peierls 1988]. The primary difference is that for sparse LU factorization, the metric can only be estimated prior to numeric factorization, which limits its use as a simple method for selecting the appropriate method.

Two questions remain: how useful is this metric, and what should the cutoff value be? We tested both methods with 320 matrices from the September 2006 version of the Collection: all symmetric positive definite matrices and all symmetric binary matrices with zero-free diagonals to which values were added to ensure positive-definiteness. The 68 random matrices listed in Table II were excluded. The relative performance of the two methods is plotted versus the flops/$|L|$ ratio, as shown in Figure 17. These results show that the flops/$|L|$ ratio is a remarkably accurate predictor of the relative performance of these two methods (much better than we

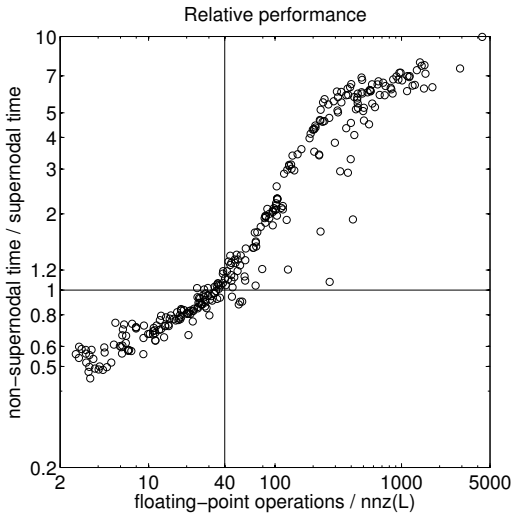


Fig. 17. CHOLMOD relative supernodal and non-supernodal performance

expected). The outliers in the plot actually strengthen the result, since it shows that most matrices fall along a smooth curve even when it is possible for any given matrix to lie far from the curve. The figure shows that a value of 40 on a Pentium 4 is a good threshold. Even when the wrong method is selected using this approach, at most a 20% performance penalty occurs for matrices in this test set. The threshold of 40 is fairly insensitive to the architecture (it would be 30 on an AMD Opteron, and 35 on a Sun Sparc). It would be impossible to determine this cutoff using random matrices or a theoretical analysis.

8. THE FUTURE

8.1 More matrices

Matrices are continually submitted and are added to the Collection every few months. Without the continual influx of new and larger matrices, the collection would become less and less useful over time. As of November 2010, we have a backlog of about three dozen matrices to be added. No one can predict the future, but we plan on continuing to augment and maintain the Collection for as many years as we can. Computational scientists are encouraged to submit their sparse matrices for inclusion in the Collection. Matrices used in published performance evaluations of sparse matrix algorithms are of particular interest, to enable repeatable experiments by other researchers. Matrices can be submitted to http://www.cise.ufl.edu/∼web-gfs, for user-name `davis`. Use a standard format for the matrix, such as a MATLAB MAT-file, a Rutherford-Boeing file, a Matrix Market file, or a list of triplets (where each line of the file contains the row index, column index, and numerical value of one entry in the matrix). Include a description of the matrix, the problem area it arises from, citations to published articles that discuss the matrix (if available), and source (in case the matrix author and submitter are different).

8.2 Better meta-data

We plan on allowing for multiple `kind` tags for each matrix, separated by semicolons. These new `kind` tags will be appended to existing matrices as well. For example, a problem currently tagged as an `economic problem` might become tagged with `kind="economic problem; least squares problem; rank-deficient"`. Each problem can include a multi-line text field called the `notes`. We include in this field the description of the problem from the submitter, citations to relevant literature, and sometimes short snippets of MATLAB code. Some of this information is available in the `README.txt` file for each matrix group. Since the `notes` field was added as an option only in October 1997, very few of the 593 problems collected before that date have a `notes` field. However, we have records of our discussions with matrix submitters, and we plan to go back to these 593 problems and include a more descriptive `note` field.

These changes to the meta-data would affect the MATLAB *.mat file and the RB and MM *.tar.gz archive files, but they would not affect the matrices themselves.

9. SUMMARY

A large, easily accessible, and actively growing collection of sparse matrices from real applications is crucial for the development and testing of sparse matrix algorithms. The University of Florida Sparse Matrix Collection meets this need, as the largest and most widely-used collection available. We have discussed our strategy for including new matrices in the Collection, which is effective although admittedly ad hoc. Five methods of searching and downloading the matrices have been presented: via MATLAB or Mathematica, via a stand-alone Java GUI program, via a standard web browser and search tool, and via Amazon Web Services^{TM}. In addition, the Collection is a valuable resource for testing and developing of graph visualization algorithms, allowing us to discover a limitation of existing coarsening schemes in handling matrices with a star-graph like substructure, as well as proposing a solution. There are still matrices that even the new coarsening scheme does not handle satisfactorily, thus the Collection continues to serve as a fertile ground for research on new algorithms, both numerical and combinatorial.

ACKNOWLEDGMENTS

We would like to thank John Gilbert for his comments on random sparse matrices and his feedback on a draft of this paper. We would like to thank Iain Duff for testing and evaluating our UFgui application, and for suggesting features that we then included in that application. We would like to thank the reviewers for their helpful comments. We would also like to thank the 359 matrix authors and 49 editors (excluding ourselves) who made this collection possible.

REFERENCES

Amestoy, P. R., Davis, T. A., and Duff, I. S. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl. 17,* 4, 886–905.

Amestoy, P. R., Davis, T. A., and Duff, I. S. 2004. Algorithm 837: AMD, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. 30,* 3, 381–388.

Amestoy, P. R., Li, X., and Ng, E. G. 2007. Diagonal Markowitz scheme with local symmetrization. *SIAM J. Matrix Anal. Appl. 29,* 1, 228–244.

Amestoy, P. R. and Puglisi, C. 2002. An unsymmetrized multifrontal LU factorization. *SIAM J. Matrix Anal. Appl. 24*, 553–569.

Bai, Z., Day, D., Demmel, J., and Dongarra, J. 1996. Test matrix collection (non-Hermitian eigenvalue problems), Release 1. Tech. rep., University of Kentucky. September. Available at ftp://ftp.ms.uky.edu/pub/misc/bai/Collection.

Bai, Z., Day, D., Demmel, J., and Dongarra, J. accessed 2008. Non-Hermitian eigenvalue problems. http://www.cs.ucdavis.edu/∼bai/NEP.

Barnes, J. and Hut, P. 1986. A hierarchical o(n log n) force-calculation algorithm. *Letters to Nature 324,* 4 (Dec.), 446–449.

Batagelj, V. and Mrvar, A. accessed 2008. Pajek networks. http://vlado.fmf.uni-lj.si/pub/networks/data.

Benzi, M. and Tuma, M. 1998. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput. 19*, 968–994.

Berger, A. J., Mulvey, J. M., Rothberg, E., and Vanderbei, R. J. 1995. Solving multistage stochastic programs using tree dissection. Tech. Rep. SOR-95-07, Dept. of Civil Eng. and Operations Research, Princeton Univ., Princeton, NJ. June.

Boisvert, R. F., Pozo, R., Remington, K., Barrett, R., Dongarra, J., Miller, B., and Lipman, B. accessed 2008. The Matrix Market. http://math.nist.gov/MatrixMarket.

Boisvert, R. F., Pozo, R., Remington, K., Barrett, R., and Dongarra, J. J. 1997. The Matrix Market: A web resource for test matrix collections. In *Quality of Numerical Software, Assessment and Enhancement*, R. F. Boisvert, Ed. Chapman & Hall, London, 125–137. (http://math.nist.gov/MatrixMarket).

Bomhof, C. W. and van der Vorst, H. A. 2000. A parallel linear system solver for circuit simulation problems. *Numer. Linear Algebra Appl. 7,* 7-8, 649–665.

Brainman, I. and Toledo, S. 2002. Nested-dissection orderings for sparse LU with partial pivoting. *SIAM J. Matrix Anal. Appl. 23*, 998–1012.

Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. 2008. Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate. *ACM Trans. Math. Softw. 35,* 3, 1–14.

Davis, T. A. 2005. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Trans. Math. Softw. 31,* 4, 587–591.

Davis, T. A. and Hager, W. W. 2009. Dynamic supernodes in sparse cholesky update/downdate and triangular solves. *ACM Trans. Math. Softw. 35,* 4, 1–23.

Demmel, J. W., Eisenstat, S. C., Gilbert, J. R., Li, X. S., and Liu, J. W. H. 1999. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl. 20,* 3, 720–755.

Demmel, J. W., Gilbert, J. R., and Li, X. S. 1999. An asynchronous parallel supernodal algorithm for sparse Gaussian elimination. *SIAM J. Matrix Anal. Appl. 20,* 4, 915–952.

Duff, I. S. 1974. Pivot selection and row ordering in Givens reductions on sparse matrices. *Computing 13*, 239–248.

Duff, I. S. 2001. YM11 documentation in HSL 2007. http://www.hsl.rl.ac.uk/specs/ym11.pdf.

Duff, I. S., Grimes, R. G., and Lewis, J. G. 1989. Sparse matrix test problems. *ACM Trans. Math. Softw. 15,* 1, 1–14.

Duff, I. S., Grimes, R. G., and Lewis, J. G. 1997. The Rutherford-Boeing sparse matrix collection. Tech. Rep. RAL-TR-97-031, Rutherford Appleton Laboratory, UK.

Duff, I. S. and Koster, J. 1999. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl. 20,* 4, 889–901.

Duff, I. S. and Pralet, S. 2005. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Anal. Appl. 27,* 2, 313–340.

Duff, I. S. and Reid, J. K. 1979. Performance evaluation of codes for sparse matrix problems. In *Performance Evaluation of Numerical Software*, L. D. Fosdick, Ed. New York: North-Holland, New York, 121–135.

Duff, I. S. and Reid, J. K. 1996. Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw. 22,* 2, 227–257.

Dumas, J.-G. accessed 2008. Sparse integer matrices collection. http://ljk.imag.fr/membres/Jean-Guillaume.Dumas/simc.html.

Feldmann, P., Melville, R., and Long, D. 1996. Efficient frequency domain analysis of large nonlinear analog circuits. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. IEEE, Santa Clara, CA.

Fruchterman, T. M. J. and Reingold, E. M. 1991. Graph drawing by force-directed placement. *Softw. Pract. Exper. 21,* 11, 1129–1164.

Gay, D. accessed 2008. NETLIB LP test problems. http://www.netlib.org/lp.

George, A., Heath, M. T., and Ng, E. 1983. A comparison of some methods for solving sparse linear least-squares problems. *SIAM J. Sci. Statist. Comput. 4,* 2, 177–187.

George, A. and Liu, J. W. H. 1981. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Gilbert, J. R., Moler, C., and Schreiber, R. 1992. Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl. 13,* 1, 333–356.

Gilbert, J. R. and Peierls, T. 1988. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM J. Sci. Statist. Comput. 9*, 862–874.

Gorter, F. and Kasky, A. 2010. The most intense moments the universe has ever known are the next 15 seconds. http://butdoesitfloat.com/496420/The-most-intense-moments-the-universe-has-ever-known-are-the-next-15.

Gould, N., Hu, Y., and Scott, J. accessed 2008. Gould/Hu/Scott collection. ftp://ftp.numerical.rl.ac.uk/pub/matrices/symmetric.

Gupta, A. 1996. Fast and effective algorithms for graph partitioning and sparse matrix ordering. Tech. Rep. RC 20496 (90799), IBM Research Division, Yorktown Heights, NY.

Gupta, A. 2002. Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices. *SIAM J. Matrix Anal. Appl. 24*, 529–552.

Hachul, S. and Jünger, M. 2004. Drawing large graphs with a potential field based multilevel algorithm. In *Proc. 12th Intl. Symp. Graph Drawing (GD '04)*. LNCS, vol. 3383. Springer, 285–295.

Higham, N. J. 2002. *Accuracy and Stability of Numerical Algorithms*, 2nd ed. SIAM, Philadelphia.

Hu, Y. F. 2005. Efficient and high-quality force-directed graph drawing. *The Mathematica Journal 10*, 37–71.

IEEE Spectrum editorial staff. 2010. The big picture: Pretty math problem. *IEEE Spectrum 47,* 10 (Oct.), 18–19.

Kamvar, S. accessed 2008. Stanford-Berkeley web matrices. http://www.stanford.edu/∼sdkamvar.

Kamvar, S. D., Haveliwala, T. H., and Golub, G. H. 2004. Adaptive methods for the computation of page rank. *Linear Algebra Appl. 386*, 51–65.

Karypis, G. and Kumar, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 359–392.

Keiter, E., Hutchinson, S. A., Hoekstra, R., Rankin, E., Russo, T., and Waters, L. 2003. Computational algorithms for device-circuit coupling. Tech. Rep. SAND2003-0080, Sandia National Laboratories, Albequerque, NM.

Kemelmacher, I. 2005. Indexing with unknown illumination and pose. In *IEEE Conf. Computer Vision and Pattern Recognition*. IEEE, San Diego.

Kiss, G. R., Armstrong, C., Milroy, R., and Piper, J. 1973. An associative thesaurus of English and its computer analysis. In *The Computer and Literary Studies*, A. Aitken, R. Bailey, and N. Hamilton-Smith, Eds. Edinburgh Univ. Press, Edinburgh.

Labarre, S. 2010. Geeky science problems double as works of art. http://www.fastcodesign.com/1662136/geeky-science-problems-double-as-works-of-art.

Leskovec, J., Chakrabarti, D., Kleinberg, J., and Faloutsos, C. 2005. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *Knowledge*

Discovery in Databases: PKDD 2005, A. Jorge, L. Torgo, P. Brazdil, R. Camacho, and J. Gama, Eds. Lecture Notes in Computer Science, vol. 3721. Springer Berlin / Heidelberg, 133–145.

McConnell, R. M. and Spinrad, J. P. 1999. Modular decomposition and transitive orientation. *Discrete Mathematics 201*, 189–241.

Mészáros, C. accessed 2008. Linear programming test set. http://www.sztaki.hu/∼meszaros/public_ftp/lptestset.

Miller, J. J. H. and Wang, S. 1991. An exponentially fitted finite element method for a stationary convection-diffusion problem. In *Computatioal methods for boundary and interior layers in several dimensions*, J. J. H. Miller, Ed. Boole Press, Dublin, 120–137.

Mittelmann, H. accessed 2008. Linear programming test set. http://plato.asu.edu/ftp/lptestset.

Ng, E. G. and Peyton, B. W. 1993. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput. 14,* 5, 1034–1056.

Page, L., Brin, S., Motwani, R., and Winograd, T. 1998. The PageRank citation ranking: bringing order to the web. Tech. rep., Stanford Digital Library Tech. Project, Stanford, CA.

Resende, M. G. C., Ramakrishnan, K. G., and Drezner, Z. 1995. Computing lower bounds for the quadratic assignment problem with an interior point algorithm for linear programming. *Operations Research 43*, 781–791.

Rudnyi, E. B. accessed 2008. Oberwolfach model reduction benchmarks. http://www.imtek.uni-freiburg.de/simulation/benchmark.

Rudnyi, E. B., van Rietbergen, B., and Korvink, J. G. 2006. Model reduction for high dimensional micro-FE models. In *Proc. 3rd HPC-Europa Transnat. Access Meeting*. HPC-Europa, Barcelona.

Ruesken, A. 2002. Approximation of the determinant of large sparse symmetric positive definite matrices. *SIAM J. Matrix Anal. Appl. 23,* 3, 799–818.

Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems*, 2nd ed. SIAM, Philadelphia.

Saad, Y. accessed 2008. SPARSKIT collection. http://math.nist.gov/MatrixMarket/data/SPARSKIT.

Schenk, O. accessed 2008. University of Basel collection. http://www.computational.unibas.ch/computer_science/scicomp/matrices.

Schulze, J. 2001. Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods. *BIT 41,* 4, 800–841.

Taylor, A. and Higham, D. J. 2009. CONTEST: A controllable test matrix toolbox for MATLAB. *ACM Trans. Math. Softw. 35,* 4, 1–17.

Trefethen, L. N. 2002. A hundred-dollar, hundred-digit challenge. *SIAM News 35,* 1 (Jan/Feb).

van Heukelum, A., Barkema, G. T., and Bisseling, R. H. 2002. DNA electrophoresis studied with the cage model. *J. Comp. Phys. 180*, 313–326.

Walshaw, C. 2003. A multilevel algorithm for force-directed graph drawing. *Journal of Graph Algorithms and Applications 7*, 253–285.

Watts, D. J. and Strogatz, S. H. 1998. Collective dynamics of small-world networks. *Nature 393*, 440–442.

Wilkinson, J. H. 1971. Linear algebra; part II: the algebraic eigenvalue problem. In *Handbook for Automatic Computation*, J. H. Wilkinson and C. Reinsch, Eds. Vol. 2. Springer-Verlag, Berlin, New York.

Zitney, S. E. 1992. Sparse matrix methods for chemical process separation calculations on supercomputers. In *Proc. Supercomputing '92*. IEEE Computer Society Press, Minneapolis, MN, 414–423.

Zitney, S. E., Mallya, J., Davis, T. A., and Stadtherr, M. A. 1996. Multifrontal vs. frontal techniques for chemical process simulation on supercomputers. *Comput. Chem. Eng. 20,* 6/7, 641–646.

Received November 2008; revised March 2010; accepted November 2010.