SAND2006-3364C

Combinatorial Scientific Computing: The Enabling Power of Discrete Algorithms in Computational Science

Bruce Hendrickson[1] and Alex Pothen[2]

[1] Discrete Math & Algorithms Dept., Sandia National Labs, Albuquerque, NM, USA
bah@sandia.gov,
WWW home page: http://www.cs.sandia.gov/∼bahendr
[2] Computer Science Department and Center for Computational Science,
Old Dominion University, Norfolk, VA, USA
pothen@cs.odu.edu,
WWW home page: http://www.cs.odu.edu/∼pothen

Abstract. Combinatorial algorithms have long played a crucial, albeit under-recognized role in scientific computing. This impact ranges well beyond the familiar applications of graph algorithms in sparse matrices to include mesh generation, computational biology and chemistry, data analysis and parallelization. Trends in science and in computing suggest strongly that the importance of discrete algorithms in computational science will continue to grow. This paper reviews some of these many past successes and highlights emerging areas of promise and opportunity.

1 Introduction

Combinatorial scientific computing (CSC) is a new name for research in an inter-disciplinary field that spans scientific computing and algorithmic computer science. Research in CSC involves three key components. The first component involves identifying a problem in scientific computing and building an appropriate combinatorial model of the problem, in order to make the computation feasible or efficient. Developing the right combinatorial model is often critical to the computation of an efficient solution, and this step could be the most time-consuming of all. The second component involves the design, analysis, and implementation of algorithms to solve the combinatorial subproblem. The emphasis in this step is on practical algorithms that are efficient for large-scale problems; an algorithm with a time complexity quadratic in the input size could be too slow to be useful, if the worst-case behavior is realized. The algorithm could compute an exact, approximate, or heuristic solution to the problem, and it should run quickly within the context of the other computational steps in the scientific computation. The third component involves developing software, evaluating its performance on a collection of test problems, making it publicly available, and perhaps integrating with a larger software library. These three components are illustrated in several examples of research activity in CSC included in this paper. Work in CSC is

multi-disciplinary in its orientation, and has the twin emphases of theoretical rigor and practical impact.

While work in CSC has been ongoing for more than three decades, the myriad roles of combinatorial algorithms are scattered in standard taxonomies of scientific computing. This historical fragmentation has obscured the broad impact of combinatorial algorithms in scientific computing. Algorithmic researchers in one niche are often unaware of ongoing work in another, perhaps related niche. Yet, a developer of computational geometry algorithms for mesh generation is likely to have more in common esthetically and intellectually with a developer of sparse matrix algorithms than with a user of the meshes he or she develops. The CSC community was founded to address this fragmentation and to facilitate closer interactions among researchers in this field.

The purpose of this article is to briefly highlight the role that combinatorial algorithms have played in various fields of scientific computing, and to point to emerging opportunities for the future. The topics discussed include the role of CSC in parallel computing; differential equations, sparse linear algebra, and numerical optimization; statistical physics, computational chemistry, bioinformatics, and information sciences.

We are aware of two articles with similar goals as ours in the related fields of scientific computing and theoretical computer science. The central role of algorithms in numerical analysis (we would call it scientific computing) has been surveyed recently by Trefethen [63]. A report on challenges for theoretical computer science, as emerging from an NSF-funded workshop circa 2000, was drafted by Johnson [35].

We view this document as a work in progress, and invite suggestions and feedback from other researchers both within and outside the CSC community.

2 Parallel Computing

In recent years, parallel computing has become central to scientific and engineering simulations. The efficient parallelization of scientific computations requires the solution to a variety of combinatorial problems. Perhaps best known is the need to partition the data (and attendant work) of a problem amongst the processors of a parallel machine. For the past decade, this problem has been commonly cast in terms of graph partitioning. Vertices of the graph represent units of computation, and the edges describe data dependencies. The goal of graph partitioning is to divide the vertices into sets of approximately equal cardinality (or weight) while cutting as few edges as possible [32]. Several widely used serial and parallel graph partitioning tools have been developed for this purpose [33, 37, 38].

Unfortunately, the number of graph edges cut by a partitioning is only an approximation to the actual communication volume in a parallel calculation. So minimizing the number of cut edges doesn't actually minimize communication. More recently, alternative *hypergraph* partitioning models has been devised in which the number of cut *hyperedges* exactly corresponds to communication vol-

ume [9]. A hypergraph is a generalization of a graph in which a *hyper*edge can connnect of two *or more* vertices. As above, vertices represent computation. But now a hyperedge joins all the vertices that consume a value with the vertex that produces it. Work is divided amongst processors by partitioning the vertices in such a way that a minimum number of hyperedges is cut. Based upon this insight, serial and parallel hypergraph partitioning tools have been developed to facilitate parallel computations [10, 18].

Graph coloring is another important kernel for parallelizing some scientific operations. A *coloring* is an assignment of a color to each vertex in such a way that adjacent vertices have different colors. The goal is to label all the vertices while using only a small number of colors. Coloring is a useful tool in parallelizing applications in which an operation on a vertex has side effects on adjacent vertices. In this case, an operation cannot be simultaneously performed on neighboring vertices. The coloring identifies sets of vertices (those with the same color) that can be operated on at the same time. These operations can all be performed in parallel. Thus, a small number of colors facilitates a fast parallel calculation. As one example of this idea, Jones and Plassmann use coloring to identify elements that can be simultaneously refined in an adaptive meshing application [36].

Parallel computing also requires efficient algorithms for interprocessor communication. This challenge leads to a set of problems that can be addressed with discrete algorithms. Consider the common situation in which each processor needs to send information to a few other processors. This is a recurring kernel in many scientific problems. The network of wires that carries messages in a parallel computer is generally sparse, but regular. If the logical communication pattern in the application can be embedded well into the physical network, then communication will be efficient. A good embedding is one in which no physical link is expected to transmit a disproportionate amount of data. Once a computation is broken into P pieces, there is freedom in the mapping from pieces to P physical processors. This freedom can be exploited to ensure that the communication patterns in the application map well onto the physical network. This problem can be described in terms of graph embeddings. Given two graphs G and H, an *embedding* is an assignment of vertices from G to vertices of H, with a corresponding assignment of edges of G to paths in H. In the parallel computing setting, the vertices of G are the work partitions and we want to assign them to processors (vertices of H). But we want to do this is a way such a way that communication operations (edges of G) don't overwhelm the network interconnect (edges of H). Graph embedding techniques provide a way to address this problem.

Combinatorial algorithms arise an a wide assortment of other parallel computing scenarios. Alternative load balancing models use space-filling curves or network flow algorithms. Graph matching is used to reduce data remapping costs in dynamic load balancing. Graph techniques are used to block or reorder operations to improve the utilization of memory hierarchies. As these and many

similar examples illustrate, combinatorial scientific computing is a thriving and critical enabler for parallel computing.

3 Mesh Generation

Many methods for solving partial differential equations require the geometric space to be decomposed into simple shapes. In scientific or engineering simulations with complex geometries, this *mesh generation* problem can be extremely challenging. In many settings, more time is spent generating the mesh than in any subsequent step of simulation and analysis.

Criteria to evaluate the quality of a mesh continue to evolve. But generally speaking, a good mesh is one with *well shaped* elements, and as few of them as possible. A well shaped element is one in which angles and lengths don't vary too much from being isotropic. (When the physics being modelled is dramatically skewed, as near surfaces in fluid flow calculations, a carefully skewed element may be desireable, and meshing routines must be adjusted appropriately.)

A rich collection of geometric algorithms are employed in mesh generation. One common technique is the use of Delaunay triangulation to produce triangular meshes in two dimensions [45, 58]. Delaunay triangularization is an elegant algorithm for joining a set of points with triangles in a such a way that unnecessarily badly shaped triangles are avoided. However, in the mesh generation problem the locations of mesh points in the interior of the object are generally unspecified. A wide variety of algorithms have been proposed to initialize and optimize point locations. These algorithms use geometric techniques like quadtrees (or oct-trees in three dimensions) and various point insertion algorithms. Near geometric boundaries, constrained triangulations may be required, which adds complexity to standard Delaunay algorithms.

Tetrahedral meshing in three dimensions is considerably more challenging. Unlike in two dimensions, three dimensional Delaunay meshes are not guaranteed to consist of only well shaped elements. A variety of heuristics have been proposed to avoid badly shaped *sliver* elements, and this continues to be an active area of research.

For some applications, quadrilateral (in 2D) or hexahedral (in 3D) elements are prefered to triangles and tetrahedron. Quad and hex meshing is quite challenging, and is considerably less mature than triangle or tetrahedral meshing. A quad or hex mesh has considerable topological structure. For instance, consider the path of quadrilateral elements that is constructed by entering an element on one side and departing it on the opposite side. In three dimensions, you can work with paths constructed in a similar manner, or sheets grown by expanding in two of the three pairs of opposing faces of an element. The topology of these structures greatly constrains the space of possible meshes and can be used to facilitate the mesh generation process [61].

4 Solving Linear Systems

4.1 Direct Methods

Matrix factorizations are at the core of modern numerical linear algebra. Solving systems of linear equations, least-squares data fitting, eigenvector and singular vector computations can all be described in terms of factoring a given matrix into a product of 'simpler' matrices, such as diagonal, triangular, or orthogonal matrices. When a matrix is sparse, i.e., there are many zero elements in it, the factors can be computed with fewer operations and reduced storage. As an instance, the Cholesky factorization of an $n \times n$ dense matrix requires $O(n^3)$ operations and $O(n^2)$ space, whereas if the matrix can be represented by a planar computational graph, the operations are bounded by $O(n^{3/2})$ and space by $O(n \log n)$. Appropriate graph models and sophisticated algorithms designed using these models are necessary to realize these gains.

The graph model for Cholesky factorization (Gaussian elimination of a symmetric positive definite matrix) was introduced by Parter [47], and further studied by Rose [55]. The appropriate graph here is the adjacency graph of the symmetric matrix that has a vertex v_i representing the ith row and column, and an undirected edge (i, j) representing each nonzero a_{ij} and its symmetric counterpart a_{ji}. In the kth step of sparse Cholesky factorization, a multiple of the kth row of the current matrix is added to higher-numbered rows in which the kth column has nonzeros, with the goal of transforming such nonzeros into zeros. During this process, a zero element in a higher-numbered row could become nonzero since the kth row has a nonzero in that column. Such newly created nonzero elements during the factorization process are called *fill elements*. Parter described the graph transformation that models the kth step of the factorization: add edges as needed to make all higher-numbered neighbors of vertex v_k a clique, and then mark v_k and all edges incident on it as deleted. The edges added correspond to fill elements. Rose showed that the *filled graph* obtained by taking the union of all the added edges with the edges corresponding to the original matrix is a *chordal graph*. (A *chord* in a cycle is an edge that joins two non-consecutive vertices on the cycle. A chordal graph has a chord joining every cycle of length greater than or equal to four.) A *fill path* in a graph is a path joining two vertices v_i and v_j in which every interior vertex in the path is numbered lower than both end vertices. Rose, Tarjan, and Lueker [56] obtained a static characterization of fill in Cholesky factorization: A fill edge (v_i, v_j) is created during the factorization if and only if a fill path joins the vertices v_i and v_j in the adjacency graph of the original matrix A.

Sparse matrix factorizations require the computation of data structures for the factor matrices, i.e., identification of the nonzero elements and their row and column indices. Once this information is available, most of the non-numerical operations can be removed from the inner-most loop of numerical computations, so that the latter can be performed in time proportional to the number of numerical operations. Hence one of the requirements for algorithms for computing the various data structures associated with sparse factorizations is that

they run as far as is possible, in time proportional to arithmetic operations or faster. This necessitates the development of graph models, efficient algorithms and high-performance implementations.

The Elimination Tree A data structure called the *elimination tree* [43] has come to be recognized to play center stage in determining the control flow and in designing efficient algorithms for computing data structures for the Cholesky factors. The elimination tree has the vertices of the adjacency graph for its nodes, and the parent of a node v is the next vertex to be eliminated from the clique created when v is eliminated. In other words, the parent of a node in the etree is the lowest-numbered node among all of its higher-numbered neighbors.

The elimination tree is a minimal representation of the control dependences in the Cholesky factorization in the following sense: if an edge (i, j) exists in the filled graph then j is an ancestor of i in the elimination tree; and if i and j belong to vertex-disjoint subtrees in the elimination tree then no edge joins i and j in the filled graph. In particular this implies that the computation of a column of the factor cannot be completed unless all of the columns corresponding to its children nodes has been computed. Hence the elimination tree can be used to schedule the numerical computations associated with Cholesky factorization.

The elimination tree is also useful in designing efficient algorithms for computing the nonzero structures of the rows and columns of the Cholesky factor. The nonzero row indices in the jth column of the Cholesky factor can be obtained by unioning the row indices of the jth column of the matrix A and the row indices of the columns of the Cholesky factor corresponding to the children of node j in the elimination tree, where we consider only rows from j to n in each column. Without the elimination tree, the union would have to be taken over the set of columns with nonzeros in the kth row of the lower triangle of the factor (equivalently all the lower-numbered neighbors of vertex j in the filled graph). The structure of the ith row of the factor can be obtained as a pruned subtree of the elimination tree rooted at node i. The pruning cuts every path from i to a leaf in the elimination tree at some node h such that the element a_{hi} is nonzero in the matrix A.

Another important feature of modern sparse factorization algorithms is the identification of dense submatrices within the sparse factors to obtain high performance on modern multiprocessors through register and cache reuse. The dense submatrices are obtained by grouping adjacent vertices with identical sets of higher-numbered neighbors into supernodes. The occurrence of supernodes stems from the fact that while a non-chordal graph could have an exponential number (in the number of vertices n) of maximal cliques, every chordal graph has at most n maximal cliques. Furthermore, the maximal cliques of a chordal graph can be organized into a tree called the clique tree, another data structure that has been used in efficient sparse matrix algorithms [7, 41].

Ordering Algorithms An important component of modern sparse matrix solvers is an algorithm that orders columns (and rows) of the initial matrix

to reduce the work and storage needed for computing the factors. These orderings also influence the effectiveness of preconditioners and the convergence of iterative solvers, and can often reduce the work needed by an order of magnitude or more. Two major classes of ordering algorithms have emerged thus far: algorithms based on the divide and conquer paradigm, exemplified by nested dissection, in which the computational graph is recursively separated into two or more connected components by removing a small set of vertices called a vertex separator at each step. The separator vertices are ordered last, and the remainder of the vertices in the graph are ordered by recursively computing separators in the subgraphs and giving them the next lower available numbers. The second class of ordering algorithms is a greedy, bottom-up algorithm that orders vertices to locally reduce fill. This class is exemplified by the minimum-degree algorithm, which chooses a vertex of minimum degree in the current graph to eliminate next.

In practice, minimum degree algorithms are implemented in a space-efficient manner such that the filled graph can be implicitly represented in the same space as the original graph, even as fill edges are created during the elimination [24]. One way to do this is to use a clique cover, i.e., a set of cliques that includes every edge in the current graph. Initially each edge in the original graph is a clique, and as a vertex is eliminated, all cliques containing that vertex are merged into a new clique. Since this union operation does not increase the size of the clique cover, we are guaranteed that the filled graph can be represented in no more space than the original graph. There is one difficulty associated with the clique cover representation though: it costs $O(n^2)$ operations to compute the degree of a vertex in the course of the ordering. Hence approximations for the degree measure which can be computed fast, in $O(n)$ time, have been developed, the most popular of which is the approximate minimum degree (AMD) algorithm [1], due to Amestoy, Davis and Duff.

Nested dissection, which was discovered by Alan George [25], has spurred much research into computing vertex separators in graphs. Important classes of graphs that occur in various applications have separators of bounded size. Planar graphs have $O(n^{1/2})$ separators [42], and this implies that systems of equations from finite element meshes of 2-dimensional problems can be solved in $O(n^{3/2})$ operations and $O(n \log n)$ space. This result explains why direct methods are often the solvers of choice for 2-dimensional problems. For 3-dimensional meshes in which each element is well shaped, the corresponding bounds are $O(n^2)$ operations and $O(n^{4/3})$ space. Spectral, geometric, and multilevel algorithms have been developed for computing separators in such graphs. Currently software for graph partitioning employ multilevel algorithms due to the good quality of the ordering and the fast computation they offer.

Ordering algorithms based on graph partitioning are better than minimum degree algorithms for reducing the work and space for large-scale problems in sparse Cholesky factorization. However, the minimum-degree algorithms compute the orderings faster. Practically, hybrid algorithms that combine these two approaches are employed. Graph partitioning is used for the top levels until the

subgraphs are small, and then the faster minimum degree orderings are computed on the small subgraphs.

Unsymmetric Problems For unsymmetric matrices, algorithms for sparse Gaussian elimination cannot neatly separate combinatorial concerns from numerical concerns as in the symmetric positive definite case. Non-symmetric or non-positive definite problems require pivoting based on the actual numerical values in the partially factored matrix for numerical stability. In these problems, the combinatorial task of data structure computation has to be interleaved with numerical computation on a group of columns.

Nevertheless, combinatorial algorithms for computing data structures for the factors and for determining the control flow have been designed. One of the important differences is that instead of an elimination tree, the control flow is determined by directed acyclic graphs (DAGs) that minimally represent the directed graphs corresponding to the factors [26]. These DAGs could be used to speed up the computation of the data structures for the factors.

Another combinatorial task that arises in these problems is matchings in graphs.An unsymmetric matrix can be represented by a bipartite graph with vertices corresponding to rows and columns, and each nonzero represented by an edge joining a row vertex and a column vertex. The magnitude of a nonzero can be represented by an edge weight, and then a matching of maximum cardinality with the maximum weight can be used to permute rows and columns so as to place large elements on the diagonal. A maximum product matching has been used in practice [19], and this reduces the need for numerical pivoting in direct solvers, and improves the quality of incomplete factorization preconditioners.

A maximum cardinality matching in a bipartite graph can also be used to compute a canonical decomposition of bipartite graphs called the Dulmage-Mendelsohn decomposition, which corresponds to a block triangular form for reducible matrices [51]. Only the diagonal blocks in a block triangular form need to be factored, potentially leading to significant savings in work and storage for reducible matrices that arise in applications such as circuit simulations. The diagonal blocks are also called strong Hall components, since they have the property that every set of k columns has nonzeros in at least $k+1$ rows. The strong Hall property is useful in many structure prediction algorithms for unsymmetric Gaussian elimination and orthogonal-triangular factorization [27].

4.2 Iterative Methods

For solving large systems of equations, iterative methods often outperform even the best sparse direct solvers. The runtime of an iterative method depends upon the cost of each iteration and the number of iterations required to achieve convergence. Combinatorial algorithms contribute to both of these considerations.

Consider the product $c = Ab$ where A is sparse. This operation typically involves a doubly-nested loop in which the outer loop is over rows and the inner loop is over nonzero entries in a row of A. If the nonzeros in a row are

not consecutive, then nonconsecutive entries of c will need to be accessed. The needed elements of c will often not be in cache, and so will be comparatively expensive to access. But if nonzeros in a row happen to be adjacent, then this will not only improve access to c, but can also be exploited to reduce the number of memory indirections required to access the elements of A.

One way to improve the performance of this operation is to reorder the columns of the matrix to increase the number of consecutive nonzeros in the rows. Pınar and Heath show that this problem can be recast as a graph problem in which the objective is to order the vertices to maximize the sum of weights connecting adjacent vertices [49]. They propose a heuristic approach to this NP-Hard problem that borrows techniques from literature on the the Traveling Salesman problem. They report that this reordering improves the performance of sparse matrix-vector multiplication by more than 20%.

The second factor in the cost of an iterative solver is the speed with which the method converges. The number of iterations can be dramatically reduced by effective preconditioning. Preconditioning is a transformation of a linear system so that $Ax = b$ is replaced by $M^{-1}Ax = M^{-1}b$, where the operator $M^{-1}A$ has better numerical properties than A alone. Generally speaking, this modification should reduce the condition number or increase the degree of clustering of the eigenvalues. For a preconditioner M to be effective, it must be easy to solve systems of the form $My = z$, and the construction of M must be efficient in both time and space.

A number of preconditioning strategies have been proposed, and several classes of preconditioners have combinatorial aspects. Incomplete factorization preconditioners follow the steps of a sparse direct solver, but discard many of the fill elements. Their construction involves many of the same operations as sparse direct solvers including graph-based reordering and fill monitoring [57]

Algebraic multigrid preconditioners approximate a matrix by a sequence of smaller and smaller matrices. The construction of smaller matrices can involve graph matching [39] or independent set computations [16].

Support theory preconditioners exploit an equivalence between the numerical properties of diagonally dominant matrices and graph embedding concepts of congestion and dilation [8]. The (symmetric positive-definite) matrix A is represented as a graph, and the preconditioner is constructed via graph operations to create an approximation to A that is easy to factor [28]. Spielman and Teng have used this approach to propose preconditioners that are provably near optimal for all diagonally dominant matrices, no matter how irregularly structured or poorly conditioned [59].

5 Optimization, Derivatives, and Coloring

5.1 Overview

Many algorithms that solve nonlinear optimization problems and differential equations require the computation of derivative matrices of vector functions.

When the derivative matrices are large and sparse, sparsity and matrix symmetry can be exploited to compute their nonzero entries efficiently. The problem of minimizing the number of function evaluations needed to compute a sparse derivative matrix can be formulated as a matrix partitioning problem.

Graph coloring is an abstraction for partitioning a set of objects into groups according to certain rules. Hence it is natural that the matrix partitioning problems in derivative matrix computations can be modeled as specialized graph coloring problems. Remarkably, the techniques for exploiting sparsity here are essentially the same whether derivatives are computed using the older method of finite differences or the comparatively recent method of automatic differentiation. In formulating, analyzing, and designing algorithms for these matrix partitioning problems, graph coloring has proven to be a powerful tool. Indeed, modern software for computing large, sparse Jacobians and Hessians rely on graph coloring algorithms to make the computations feasible.

5.2 A Jacobian Computation Problem

Let $F(x)$ denote a vector function of a vector variable x, and let J denote the derivative matrix of F with respect to x. We assume that the nonzero structure of J is known or can be computed. From the approximation $\frac{1}{\epsilon}[F(x + \epsilon e_k) - F(x)] \approx J(x)e_k$, by differencing the function along the co-ordinate vector e_k, we can estimate the kth column of J through function evaluations at $F(x)$ and $F(x + \epsilon e_k)$, where ϵ is a small step size. Thus, if sparsity is not exploited, the estimation of a Jacobian matrix with n columns would require n additional function evaluations.

Now consider a subset of the columns of the Jacobian such that no two columns have a nonzero in a common row; such a subset of columns is *structurally orthogonal*. In a group of structurally orthogonal columns, the columns are pairwise orthogonal to each other independent of the numerical values of the nonzeros. Choose a column vector d with 1's in components corresponding to the indices of columns in a structurally orthogonal group of columns, and zeros in all other components. By differencing the function F along the vector d, one can simultaneously determine the nonzero elements in *all* of these columns through the function evaluations at $F(x)$ and $F(x + \epsilon d)$. Further, by partitioning the columns of the Jacobian into the fewest groups, each consisting of structurally orthogonal columns, the number of (vector) function evaluations needed to estimate the Jacobian matrix is minimized.

Curtis, Powell, and Reid [15] observed in 1974 that sparsity can be employed in this way to reduce the number of function evaluations needed to estimate the Jacobian. In 1983 Coleman and Moré [12] modeled this matrix partitioning problem as a distance-1 graph coloring problem. The model uses the *column intersection graph* of a matrix where columns correspond to vertices and two vertices are joined by an edge whenever the corresponding columns have nonzeros in a common row (i.e., the columns are structurally non-orthogonal). A distance-1 coloring of a column intersection graph, partitions the columns into groups of structurally orthogonal columns. Since the distance-1 graph coloring problem is

known to be NP-hard, the work of Coleman and Moré showed that it is unlikely that there is a polynomial time algorithm for partitioning the columns of a matrix into the fewest groups of structurally orthogonal columns. Meanwhile, they developed several practically effective heuristics for the problem. More recently, Gebremedhin et al. [22] have used a different graph coloring model for the same matrix partitioning problem. This coloring formulation uses a *bipartite graph* to represent a Jacobian matrix. The vertex set V_1 in the bipartite graph corresponds to the rows of the matrix and the vertex set V_2 corresponds to the columns. An edge joins a row vertex r_k to a column vertex c_ℓ if the matrix element $j_{k\ell}$ of the Jacobian is nonzero.

Two columns in the Jacobian matrix are structurally orthogonal if and only if they are at a distance greater than two from each other in the corresponding bipartite graph. Thus, a distance-2 coloring of the set of column vertices V_2 is equivalent to a partitioning of the columns of the matrix into groups of structurally orthogonal columns. A distance-2 coloring of the vertex set V_2 is an assignment of colors to these vertices such that every pair of column vertices at a distance of exactly two edges from each other receives distinct colors. More precisely, this coloring is a *partial* distance-2 coloring of the bipartite graph since the row vertex set V_1 is left uncolored.

5.3 Variations on Matrix Computation

Depending on the type of derivative matrix being computed and the specifics of the method being applied, there exist several variant matrix partitioning problems. Specifically, the nature of a particular problem in our context depends on: whether the matrix to be computed is nonsymmetric, a *Jacobian*; or symmetric, a *Hessian*; whether the evaluation scheme employed is *direct* or *substitution*-based (a direct method requires solving a diagonal system and a substitution method relies on solving a triangular system of equations); whether a *unidirectional* (1d) partition or a *bidirectional* (2d) partition is used (a unidirectional partition involves only columns or rows whereas a bidirectional one involves both columns and rows); and whether *all* of the nonzero entries of the matrix or only a *subset* need to be determined; we refer to these as *full* and *partial* matrix computation.

Each of these matrix partitioning problems can be modeled as a specialized graph coloring problem.

Hessians In 1979 Powell and Toint [52] extended the approach of Curtis, Powell, and Reid to compute sparse Hessians. McCormick [44] introduced a distance-2 graph coloring model for the computation of Hessians in 1983. Independently, in 1984, Coleman and Moré [13] gave a more precise coloring model that exploits symmetry. Their model satisfies the two conditions: (1) every pair of adjacent vertices receives distinct colors (a distance-1 coloring), and (2) every path on four vertices uses at least three colors. This variant of coloring is called *star* coloring, since in such a coloring every subgraph induced by vertices assigned any two colors is a collection of stars.

Substitution-based Evaluation The evaluation scheme that we have considered is a direct one, in that each nonzero element of a derivative matrix is obtained from some row of a matrix-vector product via a finite difference operation or an AD pass. An alternative evaluation scheme would be substitution-based, in which the unknown matrix elements are determined by solving a triangular system of equations. A substitution-based evaluation is often effectively combined with the exploitation of symmetry, and hence is used in computing the Hessian.

Based on the work of Powell and Toint [52], Coleman and Moré [13] found a coloring model for a restricted substitution method for evaluating a Hessian called triangular coloring. Triangular coloring exploits symmetry only to a limited extent. A more accurate model for a substitution method to compute a Hessian leads to an *acyclic coloring* problem in which the requirements are that (1) the coloring corresponds to a distance-1 coloring, and (2) vertices in every cycle of the graph are assigned at least three distinct colors. This variant of coloring is called acyclic since every subgraph induced by vertices assigned any two colors is a forest, and is due to Coleman and Cai [11]. Recently Gebremedhin et al [23] have developed the first practical heuristic algorithm for acyclic coloring and a new efficient algorithm for star coloring; they have shown that a substitution method based on acyclic coloring leads to faster Hessian computations than a direct method based on star coloring.

Bidirectional Partition The third source for problem variation while computing the Jacobian lies in one's choice to compute it by partitioning the columns (as we have discussed earlier), or rows, or both columns and rows. If the matrix contains a few dense columns *and* rows, it may be advantageous to consider partitioning subsets of *both* columns and rows. A partition that involves only columns or rows is referred to as *unidirectional* and one that involves both columns and rows is called *bidirectional*. Due to symmetry, there is no advantage in considering a bidirectional partition of the Hessian, i.e., a symmetry-exploiting unidirectional partition suffices. In the context of automatic differentiation, bidirectional partitions arise when the Jacobian is computed by using the forward and reverse modes simultaneously.

Bidirectional partitioning of the Jacobian leads to specialized *bicoloring* problems in the bipartite graph, i.e., a coloring of subsets of both the row vertices and the column vertices with disjoint sets of colors. When bidirectional partitioning is used within a direct evaluation scheme for Jacobians, the coloring problem is that of *star bicoloring*; the corresponding model within a substitution-based scheme is the *acyclic bicoloring* problem. Bidirectional partitioning problems and their graph coloring formulations were studied by Hossain and Steihaug [34] and Coleman and Verma [14].

Partial Computation The final variation within the classification scheme is whether all elements of the Jacobian and the Hessian are required, or only a subset that would be needed for preconditioning purposes. We refer to these

	1d partition	2d partition	
Jacobian	distance-2 coloring	star bicoloring	Direct
Hessian	star coloring	NA	Direct
Jacobian	NA	acyclic bicoloring	Substitution
Hessian	acyclic coloring	NA	Substitution

Table 1. Graph coloring formulations for computing *all* nonzero entries of derivative matrices. The Jacobian is represented by its bipartite graph, and the Hessian by its adjacency graph. NA stands for not applicable.

variations as *full* and *partial* matrix computation. The latter would be useful in 'matrix-free' methods for large-scale problems, where the Jacobian is too large to be explicitly estimated, but a coarser representation of the Jacobian is used as a preconditioner. Partial matrix computation problems lead to restricted coloring problems where only a specified subset of the vertices need to be colored; however, one still needs to pay attention to the remaining vertices, since they could interfere with the estimation of the required matrix elements. Partial matrix computation problems and their coloring formulations were treated in Gebremedhin et al. [22].

All of these variations lead to a rich collection of graph coloring problems. Table 1 shows the collection of five coloring problems that arise when we consider the computation of all nonzero entries of Jacobians and Hessians. Partial matrix computation problems lead to another set of five coloring problems, of which graph models have been formulated for direct methods by Gebremedin et al.

Graph and hypergraph coloring have been used in a wide collection of application areas in addition to optimization: register allocation in compilers, radio and wireless networks, scientific computing, data movement in distributed and parallel computing, facility location problems, cache-efficient algorithms, etc. Parallel computers make it feasible to solve large-scale problems in many of these application areas, especially optimization, and hence there is currently increased interest in efficient algorithms and software for coloring graphs with millions of vertices.

6 Statistical Physics

The inherent complexity of the physical world has led physicists to investigate simplified, idealized models. The hope is that these idealized models capture some of the most interesting features of reality, but their simplification allows for more detailed analysis and simulation. In many cases, these models have rich and exploitable combinatorial structure.

The best known example of this approach is the Ising model for magnetic materials. Bulk magnetism is caused by the alignment of atomic spins. The spin of each atom influences the spin of its near neighbors, leading to very complex dynamics. In real magnets, the complex, three-dimensional geometry of atomic locations and the subtlety of the interactions makes analysis quite difficult. In

the 1930s, long before computational simulation was even an option, the Ising model was proposed as a simple tool for studying magnetism.

In the Ising model atoms are placed on the lattice points of a regular 1-, 2- or 3-dimensional grid. Each atom only interacts with its nearest neighbors. Various initial and boundary conditions can be applied to the problem, and many questions can be asked about its statistical dynamics or energetics. Some of these questions can be addressed via combinatorial optimization techniques involving matchings and counting of subgraphs [5].

The success of the Ising model has led to a vast array of variants and generalizations, many of which have combinatorial features of their own. It has also led to philosophically related models of very different phenomena.

One of these alternatives was proposed by Thorpe to model mechanical properties of materials [62]. Instead of trying to explicitly model the detailed bond structure of a complex composite, Thorpe's model places atoms in space and then connects them to near neighbors randomly. The number of connections is chosen to reflect the statistical properties of a specific composite material. These bonds are then treated as rigid bars, and the mechanical rigidity of the resulting structure can be analyzed. Fast algorithms for analyzing these structures have been developed which build upon concepts in graph matching and graph rigidity [3]. The application of combinatorial optimization techniques to simplified physical models continues to be a a very active area of research.

A very different class of idealized models of physical reality are provided by *cellular automata*, of which Conway's Game of Life is a prototypical example. In cellular automata, a set of entities interact via very simple rules, but in some circumstances complex collective behavior can be observed. The analysis of cellular automata is richly combinatorial [64].

7 Computational Chemistry

There is a natural correspondence between the structure of molecules consisting of atoms and bonds, and the vertices and edges of a graph. This relationship has led to a wide range of graph theoretic techniques in chemistry. In fact, the term "graph" as used here was first coined by J. J. Sylvester in his studies of molecular structure [60].

The graphs that describe molecules have special properties that sometimes allow for more efficient algorithms. Since an atom can be bonded to at most a few other atoms (typically four), the corresponding graph has a small maximum degree. Also, each atom is of a particular type (carbon, oxygen, etc.), so each vertex in the graph can be assigned a corresponding type value.

If a drug company discovers a molecule that exhibits an interesting biological effect, they will want to test similar molecules. But in a very large universe of molecules, how do you determine which ones are *similar*? A common technique is to use a set of graph properties or invariants to characterize molecules. Molecules of interest are then those that have similar graphical properties. This seemingly

simplistic approach actually works quite well, and a vast collection of graphical invariants have been proposed to characterize molecules [54].

Another way to search for drug candidates is to identify a piece of a molecule, perhaps a small portion of a large protein, that displays the desired activity, and then to search for other molecules that possess the same molecular fragment. This problem can phrased in terms of *subgraph isomorphism*. Given graph G and a smaller target graph T, the subgraph isomorphism problem is a search for a subset of vertices and edges in G that comprise an exact match for T. Although subgraph isomorphism is known to be NP-complete, several aspects of this application make it solvable in practice. First, the vertex types (i.e. chemical species) constrains the search space. And the goal here is not to look in a single large graph G, but rather to scan a large library of smaller graphs, each of which corresponds to a molecule.

8 Bioinformatics

Bioinformatics has seen spectacular growth since the 1990's, and combinatorial problems abound in biological applications, so that a section of this length has to be necessarily incomplete. A few books discussing algorithms in bioinformatics include: Gusfield [29], Durbin et al. [20], Pevzner [48], and Eidhammer et al. [21]. All that we hope to do here is to highlight a few areas of current research interest.

Algorithms on strings are used in local and multiple alignment of DNA and protein sequences. Dynamic programming is used to compute optimal pairwise alignments of sequences, but due to its quadratic time complexity, it is impractical for searching a large database of sequences against a query sequence to find the best local or global alignment. Faster heuristic algorithms such as BLAST and FASTA have been developed for this problem, and BLAST represents one of the most widely used bioinformatics tools. Optimal multiple sequence alignments are NP-hard to compute, and various approximation algorithms have been developed for this problem. Hidden Markov models are used to build probabilistic models of protein families and to answer queries about whether a specific protein belongs to the family or not [20]. Sequence data of specific proteins have been used to construct phylogenetic trees, and have provided an alternative to classifying organisms based on phenotypes. Constructing an optimal phylogenetic tree that represents given sequence data for multiple organisms is computationally intensive due to the super-exponential growth in the number of trees that must be examined as a function of the number of species.

Aligning RNA sequences is a computationally more intensive task since secondary structure needs to be taken into account to compute alignments. A recent approach to this problem involves a graph-theoretic formulation that uses weighted matchings in graphs and integer linear programs [4].

Proteomic experiments such as the yeast 2-hybrid system yield protein-protein interaction graphs at the organism-scale, and such graphs are now available for many model organisms as well as humans. Because these in-vitro experiments have high error rates, Bayesian networks have been used to integrate

this data with other proteomic and genomic data to improve the reliability of the interaction graph. A functional module is a group of proteins involved in a common biological process [30]. A key computational task is to decompose a protein-protein interaction graph into functional modules, to annotate the biological process that each module is involved in, and to identify "cross-talk" between the modules, i.e., the proteins that are involved in linking different biological processes. This task can be modeled as a clustering of the graph in which the clusters can overlap; computing a clustering in these networks is challenging since these graphs are small-world networks (the average distance between any two vertices in the network is $O(\log n)$, where n is the number of vertices); hence the distance between two clusters is quite small. The degree distributions of vertices in these networks obey a power-law, and there are a few vertices of high degree that tend to confound the clustering.

Early work on clustering these networks has involved searching for cliques of small size, or local clustering approaches that grow clusters from seed vertices [2]. Spectral and multi-level clustering algorithms, similar to their analogues in graph partitioning algorithms, have been developed to compute such clusterings [53]. The emerging field of computational systems biology is rich in combinatorial problems that arise from the characterization of biological networks and knowledge discovery in such networks. Effective methods to text-mine the literature to build proteomic and genomic networks is essential as the number of publications in these emerging fields continues to grow.

9 Information Processing

Like many aspects of society, science is being transformed by the explosive growth of available information and the rapidly evolving tools for search and analysis. There are many ways to represent information, but several of them have rich combinatorial underpinnings.

Perhaps the most familiar graph in informatics is the graph in which web pages are vertices and hyperlinks become (directed) edges. The structure of this graph is a critical aspect of Google's PageRank algorithm for ranking web pages [46], and of Kleinberg's related HITS algorithm [40] which is used by other search engines. Both of these algorithms construct a matrix from the web graph and then compute rankings with eigenvectors or singular vectors of this matrix.

The ideas in PageRank were actually proposed several decades earlier in citation analysis [50]. In this field, scientific papers are vertices and their citation links are edges. A calculation analogous to PageRank is used to determine which papers (or journals) are most significant.

Many types of information can be naturally represented as graphs, including social or communication networks of all sorts. The study of these *complex networks* has recently grown to become a very active area of research amongst sociologists, statistical physicists and computer scientists. Needless to say, graph algorithms are central to this field.

Text analysis is another area of informatics in which graph models are important. Consider a set of documents and the union of all their keywords. A matrix can be used to encode the set of documents that use a particular keyword. Equivalently, this relationship can be encoded by a bipartite graph in which documents are one set of vertices and keywords are another. An edge connects a document to a keyword that the document contains. This structure can be used to analyze and organize the information contained in the corpus of documents using a variety of graph and linear algebra techniques. For instance, this structure can be used in a query processing system to identify the documents that best match a user's query [17, 6, 31].

10 The Future

This paper has tried to introduce some of the many ways in which abstractions and algorithmic advances in computer science have played a role in scientific computing. The critical enabling role that these algorithms play is often overlooked. One important reason for this is that the combinatorial kernel is often just one piece of a larger tool or body of work (e.g. an ordering code within a linear solver, or a Delaunay triangulation within a mesh generator). But we believe there are other, cultural factors involved as well.

Computational science is usually marketed with an emphasis on the scientific impact of the work – e.g. the insight into global warming or the design of a more efficient chemical plant. The vast collection of enabling technologies underpinning these applications often get short shrift. When these algorithms are emphasized, those that are most accessible to computational scientists are the ones that are most likely to be lauded. The training of a computational scientist often involves exposure to numerical methods or to finite elements, so these technologies are likely to be appreciated and acknowledged. But few computational scientists have taken courses in graph algorithms, and so the importance of discrete algorithms is less likely to be recognized.

As we have tried to argue in this paper, discrete algorithms have long played a crucial enabling role in science and engineering. We expect their importance to continue to grow for several reasons. Fundamentally, as the data sets we analyze and the computations we perform continue to grow in size and complexity, optimal algorithmic efficiency becomes of paramount concern. This driving force will continue to create opportunities for new research into advanced algorithms (and approximation algorithms). In addition, as the recognition of the value of CSC becomes more widespread in the scientific and engineering communities, we are already witnessing a growing receptiveness and interest in discrete algorithms. We are also seeing a growth in educational programs that expose students to a range of topics necessary to contribute to CSC. Finally, we foresee rapid growth in several areas of science that are particularly rich in combinatorial problems. Among these are biology and informatics. And the broad transition to parallel computing for scientific and engineering computations also increases the importance of combinatorial algorithms.

As with any interdisciplanary subject, the growth of CSC raises challenges on several fronts. Education is a key issue. Computational science requires training in a scientific discipline combined with training in numerical methods and software engineering. We feel strongly that it should also include exposure to basic algorithms and data structures, with a particular focus on graph algorithms. Publication venues are a challenge in CSC since its work often falls into the cracks between theory and applications. CSC sits on the periphery of several scientific communities, but is central to none of them. Visibility and recognition is particularly important for young researchers. The tenure process can be difficult for academics whose work spans traditional communities. Professional societies and funding agencies can play an important role in nurturing and supporting this field.

Many scientific breakthroughs occur at the boundaries between fields where ideas and techniques can fruitfully cross-fertilize each other. We believe that combinatorial scientific computing lies on one of these fruitful boundaries. For researchers trained in computer science algorithms, scientific applications offer a rich assortment of interesting problems with high impact. For computational scientists trained in numerical methods or in an application discipline, combinatorial techniques offer the potential for dramatic advances in simulation capability. This mutual benefit will continue to motivate and inspire important work for long into the future.

Acknowledgments

We are endebted to the numerous colleagues who have contributed to our understanding of the diverse areas touched upon in this paper. We also thank Assefaw Gebremedhin for his comments on an earlier draft of this manuscript.

References

1. P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.
2. G. D. Bader and C. W. Hogue. An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics*, 4(2):27 pp., 2003.
3. S. Bastea, A. Burkov, C. Moukarzel, and P. M. Duxbury. Combinatorial optimization methods in disordered systems. *Computer Phys. Comm.*, 121:199–205, 1999.
4. M. Bauer, G. W. Klau, and K. Reinert. Fast and accurate structural RNA alignment by progressive Langrangian optimization. In M. R. Berthold et al, editor, *Computational Life Sciences, Lecture Notes in Bioinformatics*, volume 3695, pages 217–228. Springer Verlag, 2005.
5. R. J. Baxter. *Exactly solved models in statistical mechanics*. Academic Press, 1982.
6. M. Berry and M. Browne. *Understanding Search Engines: Mathematical Modeling and Text Retrieval*. SIAM, Philadelphia, 1999.

7. J. R. S. Blair and B. W. Peyton. An introduction to chordal graphs and clique trees. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 1–30. Springer-Verlag, 1993.
8. E. G. Boman and B. Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 25(3):694–717, 2003.
9. Ü. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, 1999.
10. Ü. Çatalyürek and C. Aykanat. PaToH a multilevel hypergraph partitioning tool for decomposing sparse matrices and partitioning VLSI circuits. Technical Report BU–CEIS–9902, Dept. Computer Engineering and Information Science, Bilkent Univ., Turkey, 1999.
11. T. Coleman and J. Cai. The cyclic coloring problem and estimation of sparse Hessian matrices. *SIAM J. Alg. Disc. Meth.*, 7(2):221–235, 1986.
12. T. Coleman and J. Moré. Estimation of sparse Jacobian matrices and graph coloring problems. *SIAM J. Numer. Anal.*, 20(1):187–209, February 1983.
13. T. Coleman and J. Moré. Estimation of sparse Hessian matrices and graph coloring problems. *Math. Program.*, 28:243–270, 1984.
14. T. Coleman and A. Verma. The efficient computation of sparse Jacobian matrices using automatic differentiation. *SIAM J. Sci. Comput.*, 19(4):1210–1233, 1998.
15. A. Curtis, M. Powell, and J. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.
16. H. De Sterck, U. M. Yang, and J. J. Heys. Reducing complexity in parallel algebraic multigrid preconditioners. *SIAM J. Matrix Anal. Appl.*, 27:1019–1039, 2006.
17. S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *J. Amer. Soc. Information Sci.*, 41(6):391–407, 1990.
18. K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek. Parallel hypergraph partitioning for scientific computing. In *Proc. IPDPS'06*. IEEE, 2006.
19. I. S. Duff and J. K. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(4):889–901, July 1999.
20. R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis*. Cambdridge University Press, 1998.
21. I. Eidhammer, I. Jonassen, and W. R. Taylor. *Protein Bioinformatics: An algorithmic approach to sequence and structure analysis*. Wiley, 2004.
22. A. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Review*, 47(4):629–705, Dec. 2005.
23. A. Gebremedhin, F. Manne, A. Pothen, and A. Tarafdar. New acyclic and star coloring algorithms with application to Hessian computations. Technical report, Old Dominion University, Norfolk, VA, March 2005.
24. A. George and J. W. H. Liu. The evolution of the minimum-degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.
25. J. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.
26. J. R. Gilbert and J. W. H. Liu. Elimination structures for unsymmetric sparse *LU* factors. *SIAM Journal on Matrix Analysis and Applications*, 14:334–352, 1993.
27. J. R. Gilbert and E. G. Ng. Predicting structure in nonsymmetric sparse matrix factorizations. In A. George, J. R. Gilbert, and J. W. H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, pages 107–139. Springer-Verlag, 1993.

28. K. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*. PhD thesis, School of Computer Science, Carnegie-Mellon University, 1996. Available as Tech. Report CMU-CS-96-123.
29. D. Gusfield. *Algorithms on Strings, Trees and Sequences*. Cambdridge University Press, 1997.
30. L. H. Hartwell, J. J. Hopfeld, and A. W. Murray. From molecular to modular cell biology. *Nature*, 402:C47–C52, 1999.
31. B. Hendrickson. Latent semantic analysis and Fiedler retrieval. *Lin. Alg. Appl.* Submitted for publication. Earlier version in Proc. SIAM Workshop on Text Mining'06.
32. B. Hendrickson and T. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26:1519–1534, 2000.
33. B. Hendrickson and R. Leland. The Chaco user's guide: Version 2.0. Technical Report SAND94–2692, Sandia National Labs, Albuquerque, NM, June 1995.
34. S. Hossain and T. Steihaug. Computing a sparse Jacobian matrix by rows and columns. *Optimization Methods and Software*, 10:33–48, 1998.
35. D. S. Johnson. Challenges for theoretical computer science: Draft. URL: `www.research.att.com/~dsj/nsflist.html`, 2000.
36. M. T. Jones and P. E. Plassmann. Parallel algorithms for adaptive mesh refinement. *SIAM J. Scientific Computing*, 18:686–708, 1997.
37. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report CORR 95–035, University of Minnesota, Dept. Computer Science, Minneapolis, MN, June 1995.
38. G. Karypis and V. Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. Technical Report 97-060, Department of Computer Science, University of Minnesota, 1997.
39. H. Kim, J. Zu, and L. Zikatanov. A multigrid method based on graph matching for convection-diffusion equations. *Numerical Lin. Alg. Appl.*, 10:181–195, 2002.
40. J. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.
41. J. G. Lewis, B. W. Peyton, and A. Pothen. A fast algorithm for reordering sparse matrices for parallel factorization. *SIAM J. Scientific Computing*, 6:1147–1173, 1989.
42. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.
43. J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.
44. S. McCormick. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Math. Program.*, 26:153–171, 1983.
45. S. J. Owen. A survey of unstructured mesh generation technology. In *Proc. 7th Intl. Meshing Roundtable*, 1998.
46. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.
47. S. V. Parter. The use of linear graphs in Gaussian elimination. *SIAM Review*, 3:119–130, 1961.
48. P. A. Pevzner. *Computational Molecular Biology: An algorithmic approach*. MIT Press, 2000.
49. A. Pınar and M. T. Heath. Improving performance of sparse matrix–vector multiplication. In *Proc. SC99*, 1999.

50. G. Pinski and F. Narin. Citation influence for journal aggregates of scientific publications: Theory, with applications to the literature of physics. *Inf. Proc. and Management*, 12:297–312, 1957.
51. A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16:303–324, Dec. 1990.
52. M. Powell and P. Toint. On the estimation of sparse Hessian matrices. *SIAM J. Numer. Anal.*, 16(6):1060–1074, 1979.
53. E. Ramadan, C. Osgood, and A. Pothen. The architecture of a proteomic network in the yeast. In M. R. Berthold et al, editor, *Computational Life Sciences, Lecture Notes in Bioinformatics*, volume 3695, pages 265–276. Springer Verlag, 2005.
54. M. Randic and J. Zupan. On interpretation of well-known topological indices. *J. Chem. Inf. Comput. Sci.*, 41(3):550–560, 2001.
55. D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In R. C. Read, editor, *Graph Theory and Computing*, pages 183–217. Academic Press, New York, 1972.
56. D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, 5:266–283, 1976.
57. Y. Saad. *Iterative methods for sparse linear systems (2nd edition)*. SIAM, Philadelphia, 2003.
58. J. Shewchuk. Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator. In M. C. Lin and D. Manocha, editors, *Applied Computational Geometry: Towards Geometric Engineering*, volume 1148 of *Lecture Notes in Computer Science*, pages 203–222. Springer-Verlag, Berlin, 1996.
59. D. Spielman and S.-H. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proc. 36th ACM Symp. Theory of Comput.* ACM, 2004.
60. J. J. Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics: With three appendices. *Amer. J. Mathematics*, 1:64–128, 1878.
61. T. G. Tautges, T. Blacker, and S. A. Mitchell. The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *Intl. J. Numerical Methods Engng.*, 39:3327–3349, 1996.
62. M. F. Thorpe. Continuous deformations in random networks. *J. Non-Cryst. Solids*, 57:355–370, 1983.
63. L. N. Trefethen. Numerical analysis. In Timothy Gowers with June Barrow-Green, editor, *Princeton Companion to Mathematics*. Princeton University Press, 2006. To appear.
64. S. Wolfram. *A new kind of science*. Wolfram Media, 2002.