Combinatorial problems in solving linear systems

Iain S. Duff[1,2], Bora Uçar[3]

[1] CERFACS,
42 Av. G. Coriolis, 31057, Toulouse, France
iain.duff@stfc.ac.uk
[2] Atlas Centre, RAL,
Oxon, OX11 0QX, England
[3] Centre National de la Recherche Scientifique,
Laboratoire de l'Informatique du Parallélisme,
(UMR CNRS-ENS Lyon-INRIA-UCBL),
Université de Lyon,
46, Allée d'Italie, ENS Lyon, F-69364, Lyon France
bora.ucar@ens-lyon.fr

Abstract. Numerical linear algebra and combinatorial optimization are vast subjects; as is their interaction. In virtually all cases there should be a notion of sparsity for a combinatorial problem to arise. Sparse matrices therefore form the basis of the interaction of these two seemingly disparate subjects. As the core of many of today's numerical linear algebra computations consists of the solution of sparse linear system by direct or iterative methods, we survey some combinatorial problems, ideas, and algorithms relating to these computations. On the direct methods side, we discuss issues such as matrix ordering; bipartite matching and matrix scaling for better pivoting; task assignment and scheduling for parallel multifrontal solvers. On the iterative method side, we discuss preconditioning techniques including incomplete factorization preconditioners, support graph preconditioners, and algebraic multigrid. In a separate part, we discuss the block triangular form of sparse matrices.

Keywords. Combinatorial scientific computing, graph theory, combinatorial optimization, sparse matrices, linear system solution

1 Introduction

In this short review paper, we examine the interplay between the solution of sparse linear systems and combinatorics. Most of this strong association comes from the identification of sparse matrices with graphs so that most algorithms dealing with sparse matrices have a close or exact analogue to an algorithm on a graph. We examine these analogues both in the case of the direct solution of sparse linear equations and their solution by iterative methods, particularly focusing on preconditioning.

Two surveys on combinatorial scientific computing have already been carried out. Hendrickson and Pothen [128] focus on the enabling role of combinatorial algorithms in scientific computing, highlighting a broad range of applications: parallel computing; mesh generation; sparse linear system solution; automatic differentiation for optimization; statistical physics; computational chemistry; bioinformatics; information processing. Bollhöfer and Schenk [29] give an overview of combinatorial aspects of LU factorization. In a spirit similar to the two preceding surveys, Heath, Ng, and Peyton [123] survey parallel algorithms for sparse Cholesky factorization by discussing issues related to the parallelization of the major steps of direct solvers. A recent book by Brualdi and Cvetković [36] covers standard matrix computations where the combinatorial tools are brought to the forefront, and graphs are used to explain standard matrix computations. The contents include matrix powers and their description using directed graphs; graph-theoretical definition of the determinant of a matrix; and the interpretation of matrix inverses and linear system solution. Brualdi and Ryser [37] and Brualdi [34,35] include a higher level of combinatorial analysis and many linear algebraic concepts beyond the solution of linear systems.

We cover linear system solution with both direct and iterative methods. We try to keep the discussion simple and provide details of some fundamental problems and methods; there are a great many beautiful results on combinatorial problems in linear algebra and reviewing them all would fill a book rather than a short survey paper. Often we review or cite the paper or papers that are at the origin of a particular method. The field has evolved in many ways and many developments have taken place since this original research. We try to provide newer references and software which define the current state-of-the-art. In some cases, survey papers of the highest quality are available, and we list some of these as pointers for readers who wish to explore these areas more fully. All the papers in our reference list are cited and almost all of them are commented on in the text of the paper. In fact we feel that the extensive bibliography is a very useful feature of this review and suggest that the reader may look at these references for further enlightenment on topics of particular interest.

We have intentionally avoided covering subareas that are addressed by other papers in this volume, for example graph partitioning, sparse matrix-vector multiplication, colouring problems, automatic differentiation.

In Section 2, we provide basic definitions from graph theory that are used throughout the paper. Some further definitions are deferred to the relevant sections. We start discussing combinatorial problems in direct solvers by a gentle introduction to the elimination process and its relationship to a suitably defined graph in Section 3. This section is structured around the main techniques that constitute the essential components of modern direct solvers. Section 4 covers some other combinatorial problems which arise in iterative methods. In this section, we mainly discuss the issues that arise due to the use of preconditioning techniques. Section 3.4 covers a special permutation of sparse matrices, known as the block triangular form, which reformulates the solution of a large linear system in terms of the solution on smaller subsystems thus giving us benefits if

the solution scheme is superlinear in the order of the system which is usually the case. We finish with some concluding remarks in Section 5.

2 Basics

In this section, we collect some elementary terms and definitions in graph theory to be used later. Most classical use of these terms and definitions in direct methods can be found in in [67,97,50]. For a more algorithmic treatment of graph theory, we refer the reader to [48].

A *graph* G is a pair (V, E), where V is a finite set, called the vertex or node set, and E is a binary relation on V, called the edge set. There are three standard graph models that are widely used in combinatorial scientific computing. In an *undirected graph* $G = (V, E)$ the edges are unordered pairs of vertices, $\{u, v\} \in E$ for $u, v \in V$ and $u \neq v$. In a *directed graph* $G = (V, E)$, the edges are ordered pair of vertices, that is, (u, v) and (v, u) are two different edges. A *bipartite graph* $G = (U \cup V, E)$ consists of two disjoint vertex sets U and V where for each edge $(u, v) \in E$ we have $u \in U$ and $v \in V$.

An edge (u, v) is said to be *incident on* the vertices u and v. For any vertex u, the vertices in the set $\text{adj}(u) = \{v : (u, v) \in E\}$ are called the *neighbours* of u. The *degree* of a vertex is the number of edges incident on it. A *path* p of length k is a sequence of vertices $\langle v_0, v_1, \ldots, v_k \rangle$ where $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. A *cycle* is a path that starts and ends at the same vertex. The two end points v_0 and v_k are said to be connected by the path p, and the vertex v_k is said to be reachable from v_0. An undirected graph is said to be *connected* if every pair of vertices is connected by a path. A directed graph is said to be *strongly connected* if every pair of vertices are reachable from each other. The *subgraph* $H = (W, F)$ of a given graph $G = (V, E)$ is a graph such that $W \subseteq V$ and $F \subseteq W \times W \cap E$. Such an H is called an *induced subgraph*, if $F = W \times W \cap E$ and a *spanning subgraph* if $W = V$. A *tree* is a connected graph without cycles. A *spanning tree* of a connected graph is a spanning subgraph which is also a tree.

Given a sparse square matrix A of order n, one can associate any of the three standard graph models described above. Formally one can associate the following three graphs. The first one is the bipartite graph $G_B = (V_R \cup V_C, E)$, where the vertex sets V_R and V_C correspond to the rows and columns of A, respectively, and the edge set E corresponds to the set of nonzeros of the matrix A so that $(i, j) \in E$ iff $a_{ij} \neq 0$. The second one is the directed graph $G_D = (V, E)$, where each vertex corresponds to a row and the respective column of A, and the edge set E corresponds to the set of nonzeros of A so that $(i, j) \in E$ iff $a_{ij} \neq 0$. The third one is the undirected graph $G_U = (V, E)$ which is defined for a pattern symmetric matrix A (that is $a_{ij} \neq 0$ whenever $a_{ji} \neq 0$), where each vertex corresponds to a row and the respective column of A, and the edge set E corresponds to the set of nonzeros so that $(i, j) \in E$ iff $a_{ij} \neq 0$ and $a_{ji} \neq 0$. We note that among these three alternatives, only the bipartite graph G_B can represent a rectangular matrix.

The three graph models discussed above suffice for the material that we cover in this paper. For completeness, we also mention three hypergraph models that are used in modeling sparse matrices. We do not cover the combinatorial scientific computing problems for which hypergraph models are used. A hypergraph $\mathcal{H}$ is a pair (V, H) where V is a finite set, called the vertex or node set, and H is a family of subsets of V, called the hyperedge set. Each hyperedge is therefore a set of vertices. Clearly, an undirected graph is a hypergraph where each hyperedge has two vertices. There are three known hypergraph models for sparse matrices. In the column-net hypergraph model [41] $\mathcal{H}_{\mathcal{R}} = (V_{\mathcal{R}}, H_{\mathcal{C}})$ of a sparse matrix matrix A, there exist one vertex $v_i \in V_{\mathcal{R}}$ and one hyperedge $h_j \in H_{\mathcal{C}}$ for each row r_i and column c_j, respectively. In this model, $v_i \in h_j$ if and only if $a_{ij} \neq 0$. In the row-net hypergraph model [41] $\mathcal{H}_{\mathcal{C}} = (V_{\mathcal{C}}, H_{\mathcal{R}})$ of matrix A, there exist one vertex $v_j \in V_{\mathcal{C}}$ and one hyperedge $h_i \in H_{\mathcal{R}}$ for each column c_j and row r_i, respectively. In this model, $v_i \in h_j$ if and only if $a_{ji} \neq 0$. In the fine-grain model [40] (see also [42]), $\mathcal{H}_z = (V_z, H_{\mathcal{RC}})$ of an $m \times n$ sparse matrix matrix A, there exist one vertex $v_{ij} \in V_z$ corresponding to each nonzero a_{ij} in matrix A. For each row and for each column there exists a hyperedge such that $H_{\mathcal{RC}}$ contains m row-hyperedges $r_1, \ldots, r_m$ and n column-hyperedges $c_1, \ldots, c_n$ where $v_{ij} \in r_i$ and $v_{ij} \in c_j$ if and only if $a_{ij} \neq 0$.

A *matching* in a graph is a set of edges such that no two are incident on the same vertex. In this paper, we will be mostly interested in matchings in bipartite graphs. A matching in the bipartite graph of a matrix A corresponds to a set of nonzeros in A no two of which are in the same row or column. An *independent set* in an undirected graph is a set of vertices no two of which are adjacent. An independent set in the undirected graph of a matrix corresponds to a square principal submatrix whose nonzeros can only be on the diagonal. A *clique* is a set of mutually adjacent vertices. A clique in the undirected graph of a matrix corresponds to a dense square principal submatrix, assuming a zero free diagonal.

3 Direct methods

We start by describing the LU decomposition, sometimes called Gaussian elimination, of a nonsingular, square sparse matrix A of order n. Although there are many variations, the basic point-wise LU decomposition proceeds in $n-1$ steps, where at step $k = 1, 2, \ldots, n-1$, the formulae

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \left(a_{ik}^{(k)} / a_{kk}^{(k)}\right) a_{kj}^{(k)} \ , \quad \text{for } i, j > k \tag{1}$$

are used to create zeros below the diagonal entry in column k. Matrices of the form $A^{(k)} = \{a_{ij}^{(k)}\}$ of order $n-k+1$ are called reduced matrices. This process leads to an upper triangular matrix U. Here, each updated entry $a_{ij}^{(k+1)}$ overwrites $a_{ij}^{(k)}$, and the multipliers $l_{ik} = a_{ik}^{(k)} / a_{kk}^{(k)}$ may overwrite $a_{ik}^{(k)}$ resulting in the decomposition $A = LU$ stored in-place. Here L is a unit lower triangular matrix, and U is an upper triangular matrix. In order for this method run to completion,

the inequalities $a_{kk}^{(k)} \neq 0$ should hold. These updated diagonal entries are called *pivots* and the operation performed using the above formulae is referred to as eliminating the variable x_k from the subsequent equations.

Suppose at step k, either of the matrix entries $a_{ik}^{(k)}$ or $a_{kj}^{(k)}$ is zero. Then there would be no update to $a_{ij}^{(k)}$. On the other hand, if both are nonzero, then $a_{ij}^{(k+1)}$ becomes nonzero even if it was previously zero (accidental cancellations due to existing values are not considered as zeros, rather they are held as if they were nonzero). Now consider the first elimination step on a symmetric matrix characterized by an undirected graph. If a_{i1} is nonzero we zero out that entry. Suppose that a_{1j} is also nonzero for some $j > 1$, then we will have a nonzero value at a_{ij} after the application of the above formulae. Consider now the undirected graph $G_U(V, E)$ of A. As $a_{i1} \neq 0$ and $a_{1j} \neq 0$, we have the edges $(1, i)$ and $(1, j)$ in E. After the elimination, the new nonzero a_{ij} will thus correspond to the edge (i, j) in the graph. Since the vertex 1 does not concern us any further (due to the condition $i, j > k$ in the formulae above), we can remove the vertex 1 from the graph, thereby obtaining the graph of the reduced matrix $A^{(1)}$ of size $(n-1) \times (n-1)$. In other words, step k of the elimination process on the matrix $A^{(k-1)}$ corresponds to removing the kth vertex from the graph and adding edges between all the neighbours of vertex k that were not connected before.

Algorithm 1 Elimination process in the graph

$G_U(V, E) \leftarrow$ undirected graph of A
for $k = 1 : n - 1$ **do**
  $V \leftarrow V - \{k\}$ ▷ remove vertex k
  $E \leftarrow E - \{(k, \ell) : \ell \in \text{adj}(k)\} \cup \{(x, y) : x \in \text{adj}(k) \text{ and } y \in \text{adj}(k)\}$

This relation between Gaussian elimination on A and the vertex elimination process on the graph of A, shown in Algorithm 1, was first observed by Parter [169]. Although it looks innocent and trivial[1], this relation was the starting point for much of what follows in the following subsections.

The following discussion is intentionally simplified. We refer the reader to [67] and [97] for more rigorous treatment.

3.1 Labelling or ordering

Consider the elimination process on the matrices shown in Fig. 1. The original ordering of the matrix A is shown on the left. The elimination process on this matrix will lead to nonzeros in the factors that are zeros in the matrix A. These new nonzeros are called *fill-in*. Indeed, the resulting matrix of factors will be full. On the other hand, the ordering obtained by permuting the first row and

[1] In 2000 at Los Alamos, Seymour V. Parter told Michele Benzi that such were the reactions he had received from the referees on his paper.

column to the end (shown on the right) will not create any fill-in. As is clearly seen from this simple example, the ordering of the eliminations affects the cost of the computation and the storage requirements of the factors.

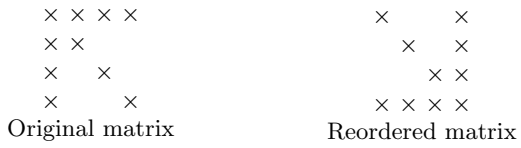

Fig. 1. Ordering affects the sparsity during elimination.

Ordering the elimination operations corresponds to choosing the pivots among combinatorially many alternatives. It is therefore important to define and find the best ordering in an efficient way. There are many different ordering methods; most of them attempt to reduce the fill-in. Minimizing the fill-in is an NP-complete problem. This was first conjectured to be true in 1976 by Rose, Tarjan, and Lueker [184] in terms of the elimination process on undirected graphs. Then Rose and Tarjan [183] proved in 1978 that finding an elimination ordering on a directed graph that gives minimum fill-in is NP-complete (there was apparently a glitch in the proof which was rectified by Gilbert [103] two years later). Finally, Yannakakis [209] proved the NP-completeness of the minimum fill-in problem on undirected graphs in 1981.

Heuristic ordering methods to reduce the fill-in predate these complexity results by about two decades. The first of these ordering methods is due to Markowitz [162]. At the beginning of the kth elimination step, a nonzero entry $a_{ij}^{(k)}$ in the reduced matrix is chosen to reduce the fill-in, and the chosen entry is permuted to the diagonal, thus defining the kth pivot. The criterion for choosing an entry $a_{ij}^{(k)}$ is to select the entry to minimize the product of the number of other entries in its row and the number of other entries in its column. Markowitz's paper deals with nonsymmetric matrices. The selection criterion was later adapted to symmetric matrices by Tinney and Walker [202] (they do not cite Markowitz's paper but state that their method might be used already). Tinney and Walker's method of choosing a diagonal entry as the pivot at step k, referred to as S2 in their paper, can be seen more elegantly during the elimination process on the graph, as noted by Rose [181]. Here, the vertex with the minimum degree in the current graph is eliminated. In other words, instead of eliminating vertex k at step k of the Algorithm 1, a vertex with minimum degree is selected as the pivot and labelled as k. Due to this correspondence, Rose renamed the method S2 of Tinney and Walker as the minimum degree algorithm.

There have been many improvements over the basic minimum degree algorithm, reducing both the run time and the space complexity. Probably the most striking result is that the method can be implemented in the same amount of space used to represent the original graph with a few additional arrays of

size n. This is surprising as the degrees changes dynamically, fill-in normally occurs throughout the execution of the algorithm, and to be able to select a vertex of minimum degree, the elimination process should somehow be simulated. The methods used to achieve this goal are described in [73,95,96]. The survey by George and Liu [98] lists, inter alia, the following improvements and algorithmic follow-ups: *mass eliminations* [100], where it is shown that, in case of finite-element problems, after a minimum degree vertex is eliminated a subset of adjacent vertices can be eliminated next, together at the same time; *indistinguishable nodes* [97], where it is shown that two adjacent nodes having the same adjacency can be merged and treated as one; *incomplete degree update* [83], where it is shown that if the adjacency set of a vertex becomes a subset of the adjacency set of another one, then the degree of the first vertex does not need to be updated before the second one has been eliminated; *element absorption* [74], where based on a compact representation of elimination graphs, redundant structures (cliques being subsets of other cliques) are detected and removed; *multiple elimination* [150], where it was shown that once a vertex v is eliminated, if there is a vertex with the same degree that is not adjacent to the eliminated vertex, then that vertex can be eliminated before updating the degree of the vertices in adj(v), that is the degree updates can be postponed; *external degree* [150], where instead of the true degree of a vertex, the number of adjacent and indistinguishable nodes is used as a selection criteria. Some further improvements include the use of compressed graphs [11], where the indistinguishable nodes are detected even before the elimination process and the graph is reduced, and the extensions of the concept of the external degree [51,105]. The *approximate minimum degree* as described in [6] is shown to be more accurate than previous degree approximations and leads to almost always faster execution with an ordering often as good as or better than minimum degree. Assuming a linear space implementation, the run-time complexity of the minimum degree (and multiple minimum degree) and the approximate minimum degree ordering heuristics are shown to be, respectively, $\mathcal{O}(n^2m)$ and $\mathcal{O}(nm)$ for a graph of n vertices and m edges [124].

A crucial issue with the minimum degree algorithm is that ties arise while selecting the minimum degree vertex [66]. It is still of interest, though a little daunting, to develop a tie-breaking strategy and beat the current fill-reducing algorithms.

As mentioned above, the minimum degree based approaches order the matrix by selecting pivots using the degree of a vertex without any reference to later steps of the elimination. For this reason, the general class of such approaches are called *local strategies*. Another class, called *global strategies*, permute the matrix in a global sense so as to confine the fill-in within certain parts of the permuted matrix. A widely used and cited algorithm is by Cuthill and McKee [49]. A structurally symmetric matrix A is said to have *bandwidth* $2m+1$, if m is the smallest integer such that $a_{ij} = 0$, whenever $|i-j| > m$. If no interchanges are performed during elimination, fill-in occurs only within the band. The algorithm is referred to as CM and is usually based on a breadth-first search algorithm. George [102]

found that reversing the ordering found by the CM algorithm effectively always reduces the total storage requirement and the arithmetic operations when using a variant of the band-based factorization algorithm (a rigorous treatment and analysis of these two algorithms is given in [158]). This algorithm is called reverse Cuthill-McKee and often referred to as RCM.

Another global approach that received and continues to receive considerable attention is called the *nested dissection* method, proposed by George [93] and baptized by Birkhoff (acknowledged in George's paper). The central concept is a *vertex separator* in a graph: that is a set of vertices whose removal leaves the remaining graph disconnected. In matrix terms, such a separator corresponds to a set of rows and columns whose removal yields a block diagonal matrix after suitable permutation. Permuting the rows and columns corresponding to the separator vertices last, and each connected component of the remaining graph consecutively results in the *doubly bordered block diagonal* form

$$\begin{pmatrix} A_{11} & & A_{1S} \\ & A_{22} & A_{2S} \\ A_{S1} & A_{S2} & A_{SS} \end{pmatrix} . \tag{2}$$

The blocks A_{11} and A_{22} can be further dissected using the vertex separator of the corresponding graphs and can themselves be permuted into the above form, resulting in a nested dissection ordering. Given such an ordering, it is evident that fill-ins are confined to the blocks shown in the form.

A significant property of nested-dissection based orderings is that they yield asymptotically optimal fill-in and operation counts for certain types of problems. It was shown in [93] that, for a matrix corresponding to a regular finite-element mesh of size $q \times q$, the fill-in and operation count using a nested dissection ordering are $\mathcal{O}(q^2 \log_2 q)$ and $\mathcal{O}(q^3)$, respectively. For a three dimensional mesh of size $q \times q \times q$, the bounds are $\mathcal{O}(q^6)$ for the operation count and $\mathcal{O}(q^4)$ for the fill-in, see also [60] for a detailed analysis. George [93] shows the asymptotic optimality of the operation count but not the fill-in. The asymptotic results were settled thoroughly in [132]. Further developments for finite-element meshes include automatic generation of nested dissection on square meshes with $q = 2^l - 1$, with l integer, in [185], and methods for arbitrary square meshes and irregular shaped regions in [66]. In [94], a heuristic is presented to perform nested dissection on general sparse symmetric matrices. The nested dissection approach was then generalized so that it yields the same bounds for systems defined on planar and almost planar graphs [148]. The generalization essentially addresses all $n \times n$ systems of linear equations whose graph has a bounded separator of size $n^{1/2}$. The results of [149] are used to obtain separators and bounds on separator sizes on planar graphs, and therefore the asymptotic optimality results apply to planar or almost planar graphs, a general class of graphs which includes two dimensional finite-element meshes. Gilbert and Tarjan [108] combine and extend the work in [94] and [148] to develop algorithms that are easier to implement than the earlier alternatives and have smaller constant factors. In [108] asymptotic optimality results are demonstrated on planar graphs, two-dimensional finite-element graphs, graphs of bounded genus, and graphs of bounded degree with

$n^{1/2}$-separators (note that without the bounded degree condition, the algorithm can be shown not to achieve the bound on fill-in).

It is not much of a surprise that hybrid fill-reducing ordering methods (combining the minimum degree and nested dissection heuristics) have been developed. We note that the essential ideas can be seen already in [202]. Tinney and Walker suggest that if there is a natural decomposition of the underlying network, in the sense of (2), then it may be advantageous to run the minimum degree algorithm on each subnetwork. The first formalization of the hybrid approach was, however, presented in [101]. In this work, the hybrid method is applied to finite-element meshes, where first a few steps of nested dissection are applied before ordering all entries (a precise recommendation is not given). The remaining entries are ordered using bandwidth minimization methods such as CM and RCM. Liu [154] uses a similar idea on general symmetric sparse matrices. Probably this is the first paper where minimum degree based algorithms are called bottom-up approaches and the separator based, nested dissection algorithms are called top-down approaches, thus defining the current terminology. Liu terminates the nested dissection earlier (up to 5 levels of dissections are applied; but this depends on the size of the graphs and there is no precise recommendation for a general problem), and then orders the remaining vertices with minimum degree, including the separator vertices in the degree counts but ignoring them during vertex selection (this method is known as *constrained minimum degree* or *minimum degree with constraints*). The merits of the proposed algorithm are listed as a reduced sensitivity to the initial ordering of the matrix and an ordering algorithm more appropriate for parallel factorization.

As we stated before, a nested dissection ordering gives asymptotically optimal storage and operation counts for square grids. For rectangular grids, however, this is not the case. It has been shown that for rectangular grids with a large aspect ratio, nested dissection is inferior to the minimum degree ordering [14], and even to the natural ordering [25]. The main issue, as stated by Ashcraft and Liu [14], is that the ordering of the separator vertices found at different dissection steps is important. For rectangular grids, Bhat et al. [25] propose to order the separator vertices in the natural ordering to minimize the profile after partitioning the rectangular grid into square sub-grids each of which is ordered using nested dissection. Ashcraft and Liu [14] develop this idea to propose a family of ordering algorithms. In these algorithms, a partitioning of the graph is first found by using either the elimination tree or by recursive application of graph bisection [13]. Then each part is ordered using constrained minimum degree. The Schur complement of the domains is formed symbolically and used to reorder the separator vertices using multiple minimum degree. Hendrickson and Rothberg [129] (concurrently with Ashcraft and Liu) and Schulze [194] develop similar algorithms. Ashcraft and Liu find multisectors instead of performing recursive bisection; Hendrickson and Rothberg and Schulze use multilevel graph partitioning; Schulze proposes an elegant coarsening algorithm.

Not surprisingly, the current state-of-the-art in fill-reducing ordering methods is based on hybrid approaches of the kind outlined above. The efficiency of these

methods is due to developments in graph partitioning methods such as efficient algorithms for computing eigenvectors to use in partitioning graphs [17,174]; and the genesis of the multilevel paradigm [38,127] which enables better use of vertex-move-based iterative refinement algorithms [87,140]. These developments are neatly incorporated in graph partitioning and ordering software packages such as Chaco [126], MeTiS [139], SCOTCH [170], and WGPP [116]. These libraries usually have a certain threshold (according to F. Pellegrini, around 200 vertices seems to be a common choice) to terminate the dissection process and to switch to a variant of the minimum degree algorithm.

3.2 Matching and scaling

As discussed before, for the elimination process to succeed, the pivots $a_{kk}^{(k)}$ should be nonzero. This can be achieved by searching for a nonzero in the reduced matrix and permuting the rows and columns to place that entry in a diagonal position. Such permutations are called *pivoting* and guarantee that an $a_{kk}^{(k)} \neq 0$ can be found for all k so long as the original matrix is nonsingular. In *partial pivoting*, the search is restricted to the kth column. A general technique used to control the growth factor is to search the column for a maximum entry or to accept an entry as pivot so long as it passes certain numerical tests. These pivoting operations are detrimental to the fill-reducing orderings discussed in the previous section, as those ordering methods assume that the actual numerical elimination will follow the ordering produced by the symbolic elimination process on the graph.

Suppose that the diagonal entries of the matrix are all nonzero. Assuming no exact cancellation, all pivots will be nonzero when they are taken from the diagonal. Notice that any symmetric permutation of the original matrix keeps the set of diagonal entries the same, and hence the fill-reducing orderings of the previous section are applicable in this case. Our purpose in this section is to summarize the methods to find permutations that yield such diagonals.

As is clear, we are searching for n nonzeros in an $n \times n$ matrix A no two of which are in the same row or column. As we mentioned earlier, this corresponds to finding a perfect matching in the bipartite graph representation of A. The existence of such a set of n nonzeros (i.e., a perfect matching in the bipartite graph) is guaranteed to exist if, for $k = 1, 2, \ldots, n$ any k distinct columns have nonzeros in at least k distinct rows—a result shown by P. Hall [122].

A few definitions are in order before describing bipartite matching algorithms. Given a matching $\mathcal{M}$, a vertex is said to be *matched* if there is an edge in the matching incident on the vertex, and to be *unmatched* otherwise. An $\mathcal{M}$-*alternating path* is a path whose edges are alternately in $\mathcal{M}$ and not in $\mathcal{M}$. An alternating path is called an *augmenting path*, if it starts and ends at unmatched vertices. The *cardinality of a matching* is the number of edges in it. We will be mostly interested in matchings of maximum cardinality. Given a bipartite graph G and a matching $\mathcal{M}$, a necessary and sufficient condition for $\mathcal{M}$ to be of maximum cardinality is that there is no $\mathcal{M}$-augmenting path in G [23, Theorem 1]—for the curious reader the second theorem of Berge gives a similar condition

for minimum vertex covers. Given a matching $\mathcal{M}$ on the bipartite graph of a square matrix A, one can create a permutation matrix M such that $m_{ji} = 1$ iff row i and column j are matched in $\mathcal{M}$. Then, the matrix AM has a zero-free diagonal. It is therefore convenient to abuse the notation and refer to a matching as a permutation matrix.

The essence of bipartite cardinality matching algorithms is to start with an empty matching and then to augment it until no further augmentations are possible. The existing algorithms mostly differ in the way the augmenting paths are found and the way the augmentations are performed. In [121] a breadth-first search is started from an unmatched row vertex to reach an unmatched column vertex. The time complexity is $\mathcal{O}(n\tau)$, where τ is the number of nonzeros in the matrix. The algorithm in [62,63], known as MC21, uses depth-first search where, before continuing the depth-first search with an arbitrary neighbour of the current vertex, all its adjacency set is scanned to see if there is an unmatched vertex. This is called a cheap assignment and helps reduce the run time. The time complexity is $\mathcal{O}(n\tau)$, but it is observed to run usually much faster than that bound. Depth-first search is also used in [143] with a complexity of again $\mathcal{O}(n\tau)$. Hopcroft and Karp [133] find a maximal set of shortest augmenting paths using breadth-first search and perform the associated augmentations at the same time. With a detailed analysis of the possible length and number of such augmentations, they demonstrate a complexity of $\mathcal{O}(\sqrt{n}\tau)$. Building upon the work of Hopcroft and Karp, Alt et al. [5] judiciously combine depth- and breadth-first searches to further reduce the complexity to $\mathcal{O}(\min\{\sqrt{n}\tau, n^{1.5}\sqrt{\tau/\log n}\})$.

Not all matrices have perfect matchings. Those that have a perfect matching are referred to as structurally nonsingular, or structurally full rank, whereas those that do not have a perfect matching are referred to as structurally singular, or structurally rank deficient. The maximum cardinality of a matching is referred to as the structural rank which is at least as large as the numerical rank.

Although permuting a matching to the diagonal guarantees existence of the pivots, it does not say anything about their magnitudes. In order to control the growth factor, it may still be necessary to perform pivoting during the course of the elimination. It is known that for diagonally dominant matrices, pivoting on the grounds of numerical stability is not necessary. Therefore, if we find a matching which guarantees diagonal dominance after a permutation and probably some scaling, we can avoid numerical pivoting. Unfortunately not all matrices can be permuted to such a form but trying to do so is likely to reduce the need for numerical pivoting. These were the motivating ideas in [69] and [167], where such an attempt is formulated in terms of *maximum weighted bipartite matchings*.

In matrix terms, Olschowka and Neumaier [167] and Duff and Koster [69] find a permutation matrix (and hence a perfect matching) M such that the product of the diagonal of the permuted matrix, $\prod \mathrm{diag}(AM)$, is maximum (in magnitude) among all permutations. Although the product form of the variables is intimidating, a simple transformation by changing each entry of the matrix to the logarithm of its magnitude reduces the problem to the well known maximum weighted bipartite matching problem. In particular, maximizing $\prod \mathrm{diag}(AM)$ is

equivalent to maximizing the diagonal sum given by $\sum \mathrm{diag}(\hat{C}M)$ for $\hat{C} = (\hat{c}_{ij})$ where

$$\hat{c}_{ij} = \begin{cases} \log |a_{ij}|, & \text{if } a_{ij} \neq 0 \\ -\infty, & \text{otherwise} , \end{cases}$$

or, to minimizing the diagonal sum given by $\sum \mathrm{diag}(CM)$ for $C = (c_{ij})$ where

$$c_{ij} = \begin{cases} \log \max_i |a_{ij}| - \log |a_{ij}|, & \text{if } a_{ij} \neq 0 \\ \infty, & \text{otherwise} . \end{cases}$$

The literature on the minimum weighted matching problem is much larger than that on the cardinality matching problem. A recent book lists 21 algorithms [39, p.121] from years 1946 to 2001 and provides codes or a link to codes of eight of them (http://www.assignmentproblems.com/). The best strongly polynomial time algorithm is by Fredman and Tarjan [88] and runs in $\mathcal{O}(n(\tau + n \log n))$. In addition to those 21 algorithms mentioned in the book above, we mention Duff and Koster's implementation of the matching algorithm, now known as MC64 and available as an HSL subroutine (http://www.hsl.rl.ac.uk/). MC64 was initially designed for square matrices, but the latest version extends the algorithm to rectangular matrices. The running time of MC64 for the maximum weighted matching problem is $\mathcal{O}(n(\tau + n) \log n)$. MC64 also provides algorithms for a few more bipartite matching problems. There are also recent efforts which aim to develop practical parallel algorithms for the weighted matching problem. In [180] and [77] parallel algorithms for maximum weighted bipartite matching are proposed. Although these algorithms are still being investigated, one cannot expect them to be entirely successful for all problems (given that depth-first search is inherently sequential [178] and certain variations of breadth-first search are also inherently sequential [110]); nevertheless, there are many cases where each algorithm delivers solutions in quite reasonable time with quite reasonable speed-ups. There are also efforts in designing parallel approximate matching algorithms [120,161,172].

We say a few words about the pioneering work of Kuhn in maximum weighted weighted matchings [143] for two reasons. Firstly, his paper was selected as the best paper of Naval Research Logistics in its first 50 years and was reproduced as [144] (there is a delightful history of the paper by Kuhn himself [145]). Secondly, it forms the basis for the algorithms [69,167] that combine matchings with matrix scaling for better numerical properties during elimination.

By linear programming duality, it is known [143] that M is a maximum weighted matching if and only if there exist dual variables u_i and v_j with

$$\begin{cases} u_i + v_j \leq c_{ij} & \text{for } (i,j) \in E \setminus M \\ u_i + v_j = c_{ij} & \text{for } (i,j) \in M \end{cases}$$

For such u_i and v_j, setting

$$D_1 = \mathrm{diag}(e^{u_i}) \quad \text{and} \quad D_2 = \mathrm{diag}(e^{v_j} / \max_i |a_{ij}|)$$

scales the matrix so that D_1AMD_2 has all ones on the diagonal and all other entries are less than or equal to one, see [69,167]. Off-diagonal entries can be one in magnitude (this can happen for example when there is more than one maximum weighted matching), but otherwise the combined effect is such that the resulting matrix has larger entries on the diagonal. If the given matrix was obtained from a strongly diagonally dominant matrix or a symmetric positive definite matrix by permutations, the maximum product matching recovers the original diagonal [167]. Therefore, it is believed that the combination of the matching and the associated scaling would yield a set of good pivots. This was experimentally observed but has never been proved.

3.3 Elimination tree and the multifrontal method

In this section, we describe arguably the most important graphical representation for sparse matrices: the elimination tree. We discuss its properties, construction and complexity, and illustrate the flexibility of this model. We then consider one of the most important elimination-tree based class of direct methods: the multifrontal method. We indicate how we can modify the tree for greater efficiency of the multifrontal method and show how it is used in a parallel and out-of-core context.

Elimination tree The elimination tree is a graphical model that represents the storage and computational requirements of sparse matrix factorization. The name elimination tree was first used in [64] and was principally used there as a computational tree to guide the factorization process. The term was also used by Jess and Kees [136], who again used the elimination tree to represent the computational dependencies in order to exploit parallelism. We note that Jess and Kees used the elimination tree of a triangulated graph. That is, they consider the graph that includes the fill-in edges obtained using a fill-reducing ordering. The formalized definitions of the elimination tree and the use of the data structures resulting from it for efficient factorization and solution are given by Schreiber [193]. Liu considers the elimination tree in some detail and provides a detailed discussion on it in [151] and [155]. The latter paper is a comprehensive survey of the elimination tree structure. Here we provide a few properties of the elimination tree and comment on its computation, mostly following the exposition in [155].

The elimination tree essentially relates to the factorization of irreducible, pattern symmetric matrices. However, some modern solvers such as MUMPS [7] extend the use of the elimination tree to unsymmetric systems by using a tree based on the structure of the matrix $|A|+|A^T|$. This extends the benefits of the efficient symmetric symbolic factorization to the unsymmetric case, and so the following discussion can apply to general LU factorization. Note that there is a rich literature on the use of directed graphs and alternative tree models for the unsymmetric case which we do not cover—the reader is referred to [51,104,106,56,117] for a the use of directed graph models in sparse factorization and to [84,85] for alternative tree models for unsymmetric matrices.

We first give a few definitions before listing some important properties of the elimination tree. *Depth-first search* (DFS) of a graph starts with an initial node v, marks the node as visited and then recursively visits an unvisited vertex which is adjacent to v. The edges that are traversed to explore an unvisited node form a tree, which is called a *depth-first search tree* [200]. Let A be a symmetric positive definite matrix having the factorization LL^T. Let G_F represent the graph of the filled in matrix, i.e., the undirected graph of $L + L^T$. Then the elimination tree $T = (V, E)$ is a depth-first search tree of the undirected graph G_F. To us, this last statement summarizes most of the structural information relating to the factorization process. Therefore we discuss this correspondence a little more. As discussed in Section 2, the ith vertex of G_F corresponds to the ith row/column of $L+L^T$. Then, running the DFS starting from the vertex n of G_F and during the recursive calls from a vertex to its unvisited neighbors, always choosing the one with the highest index first yields the elimination tree as its search tree. We note that if the matrix is reducible the first call from the vertex n will not explore all the vertices; when this happens, the highest numbered unvisited vertex should be taken as the starting vertex of another DFS. In a reducible matrix, this will yield the forest of elimination trees for each block.

We now give the most significant characteristics of the elimination tree. Firstly, the elimination tree is a spanning tree of the graph corresponding to the filled in matrix. Secondly, the elimination tree can be constructed by making an edge from each vertex $j = 1, \ldots, n-1$ to the first nonzero l_{ij} in column j so that vertex i is the parent of vertex j. As there are no *cross edges* (a cross edge is an edge of the graph but not the DFS tree and it connect vertices which do not have an ancestor-descendant relationship in the DFS tree) with respect to a DFS tree of an undirected graph, the edges of G_F that are not in the tree T are *back edges*, i.e., they are from a vertex v to another vertex in the unique path joining v to the root (see [200] and [48, Section 23.3]). Combining with the fill-path theorem of Rose et al. [184, Lemma 4, p.270], we have that for $i > j$, the entry l_{ij} is nonzero if and only if there exists a path $v_j, v_{p1}, \ldots, v_{pt}, v_i$ in the graph of A such that the vertices $v_j, v_{p1}, \ldots, v_{pt}$ are all in the subtree of the elimination tree T rooted at node v_j. Another important property characterizing the fill-in is that $l_{ij} \neq 0$, if and only if the vertex v_j is an ancestor of some vertex v_k in the elimination tree T, where $a_{ik} \neq 0$ (see [151, Theorem 2.4]).

As is evident from the previous discussion, the elimination tree depends on the ordering of elimination operations. In particular, a node (that is the associated variable) can only be eliminated after all of its descendants have been eliminated—otherwise the structure is lost and the tree no longer corresponds to the depth-first search tree of the filled-in matrix anticipated at the beginning. A *topological ordering* of the nodes of a tree refers to an ordering in which each node is ordered before its parent. It was known earlier (see, e.g., [75] and comments in [151, p.133]) that all topological orderings of the elimination tree are equivalent in terms of fill-in and computation; in particular any postorderings of the tree are equivalent. Liu [153] investigates a larger class of equivalent orderings obtained by tree restructuring operations, refereed to as *tree rotations*.

Liu combines a result of Rose [181, Corollary 4, p.198] and one of his own [151, Theorem 2.4, p.132] to note that for any node v in the tree there is an equivalent ordering in which the nodes adjacent (in the graph of A) to the nodes in the subtree rooted at v are numbered last. Using a result of Schreiber [193, Proposition 5, p.260], Liu identifies a node y as eligible for rotation if the following is true for all ancestor z (in the elimination tree) of y but not for the node y itself: the ancestors of z forms a clique in the filled graph and are connected (in the original graph, not in the tree) to the subtree rooted at z. Then, the nodes are partitioned into three sets and numbered in the following order: those that are not ancestors of the selected node while retaining their relative order; those that are ancestors of the selected node but not connected to (in the original graph, not in the tree) the subtree rooted at the selected node; then finally the remaining nodes. The final ordering will correspond to a new elimination tree in which the subtree rooted at the selected node would be closer to the root of the new tree.

Liu [151] (also in the survey [155]) provides a detailed description of the computation of the elimination tree. Firstly, an algorithm is given in which the structure of row i of L is computed using only A and the parent pointers set for the first $i-1$ nodes. This algorithm processes the rows of in turn and runs in time proportional to the number of nonzeros in L. At row i, the pattern of the ith row of L is generated; for this the latest property of the elimination tree that we mention above (see [151, Theorem 2.4]) is used. Then for for each k for which $l_{ik} \neq 0$ and the parent k is not set, the parent of k is set to be i. In order to reduce the time and space complexity, Liu observes that parent pointers can be set using the graph of A and repeated applications of set operations for the disjoint set union problem [201]. A relatively simple implementation using these operations reduces the time complexity to $\mathcal{O}(\tau\alpha(\tau, n))$, where τ is the number of entries in A and $\alpha(\tau, n)$ is the two parameter variation of the inverse of the Ackermann function—$\alpha(\tau, n)$ can be safely assumed to be less than four.

Multifrontal method Based on techniques developed for finite-element analysis by Irons [135] and Speelpenning [196], Duff and Reid proposed the multifrontal method for symmetric [74] and unsymmetric [76] systems of equations. Liu [156] provides a good overview of the multifrontal method for symmetric positive definite matrices.

The essence of a frontal method is that all elimination operations are performed on dense submatrices (called *frontal matrices*) so that these operations can be performed very efficiently on any computer often by using the Level 3 BLAS [59]. The frontal matrix can be partitioned into a block two by two matrix where all variables from the (1,1) block can be eliminated (the variables are called fully summed) but the Schur complement formed by the elimination of these on the (2,2) block cannot be eliminated until later in the factorization. This Schur complement is often called a *contribution block*.

The multifrontal method uses an *assembly tree* to generalize the notion of frontal matrices and to allow any fill-reducing ordering. At each node of the

assembly tree, a dense frontal matrix is constructed from parts of the original matrix and the contribution blocks of the children—this process is called the *assembly* operation. Then, the fully summed variables are eliminated (factorization of the (1,1) block takes place) and the resulting contribution block is passed to the parent node for assembly into the frontal matrix at that node. Clearly, one can only perform the eliminations at a node when all the contributions have been received from its children. Liu [156] defines a tree (associated with LL^T factorization of a matrix A) as an assembly tree if for any node j and its parent p, the off-diagonal structure of the jth column of L is a subset of the structure of the pth column of L and $p > j$. It is easy to see that the elimination tree satisfies this condition and that there are other tree structures which will exhibit the same property. In this setting, the contributions from j (or any descendant of it in the assembly tree) can be accommodated in the frontal matrix of p and hence the generality offered by the assembly tree might be used for reducing the memory requirements [156, p.98].

Since the elimination tree is defined with one variable (row/column) per node, it only allows one elimination per node and the (1,1) block would be of order one. Therefore, there would be insufficient computation at a node for efficient implementation. It is thus advantageous to combine or amalgamate nodes of the elimination tree. The amalgamation can be restricted to avoid any additional fill-in. That is, two nodes of the elimination tree are amalgamated only if the corresponding columns of the L factor have the same structure. As even this may not give a large enough (1,1) block, a threshold based amalgamation strategy can be used in which the columns to be amalgamated are allowed to have a certain number of discrepancies in their patterns, introducing logical zeros. Duff and Reid do this in their original paper [74] where they amalgamate nodes so that a minimum number of eliminations are performed at each node of the resulting tree. That is, they make the (1,1) blocks at least of a user-defined order. Ashcraft and Grimes [12] investigate the effect of this relaxation in amalgamation and provide new algorithms. Firstly, the notion of a fundamental supernode is defined. A fundamental supernode is a maximal chain $(n_1, n_2, \ldots, n_p)$ of nodes in the tree such that each n_i is the only child of n_{i+1}, for $i = 1, \ldots, p-1$, and the associated column structures are perfectly nested. Then, the fundamental supernodes are visited in an ordering given by a postorder, and the effect of merging children with a parent node on the number of logical zeros created is taken into account to amalgamate nodes. In [157], an efficient algorithm which determines the fundamental supernodes in time proportional to the number of nonzeros in the original matrix (avoiding the symbolic factorization altogether) is presented.

The simple tree structure of the computations helps to identify a number of combinatorial problems, usually relating to scheduling and task mapping for efficient memory use, in out-of-core solution and in parallel computing contexts. In the rest of this section, we discuss some of the issues relating to efficient memory use and parallelization.

In a multifrontal method, the pattern of memory use permits an easy extension to out-of-core execution. In these methods, memory is divided into two parts. In the static part, the computed factors of the frontal matrices are stored. This part can be moved to secondary storage. The second part, called the active memory, contains the frontal matrix being factorized and a stack of contributions from the children of still uneliminated nodes. Liu [152] minimizes the size of the active memory by rearranging the children of each node (hence creating an equivalent ordering) in order to minimize the peak active memory for processing the whole tree. In a series of papers, Agullo et al. [2,3] and Guermouche and L'Excellent [115] develop these ideas of Liu [152] concerning the assembly and partial assembly of the children and their ordering. In these papers, algorithmic models that differentiate between the concerns of I/O volume and the peak memory size are developed, and a new reorganization of the computations within the context of an out-of-core multifrontal method are presented.

George et al. [99] propose the *subtree-to-subcube* mapping to reduce the communication overhead in parallel sparse Cholesky factorization on hypercubes. This mapping mostly addresses parallelization of the factorization of matrices arising from a nested dissection based ordering of regular meshes. The essential idea is to start from the root and to assign the nodes of the amalgamated elimination tree in a round robin-like fashion along the chains of nodes and to divide the processors according to the subtrees in the elimination tree and then recursively applying the idea in each subtree. The method reduces the communication cost but can lead to load imbalance if the elimination tree is not balanced. Geist and Ng [92] improve upon the subtree-to-subcube mapping to alleviate the load imbalance problem. Given an arbitrary tree, Geist and Ng find the smallest set of subtrees such that this set can be partitioned among the processors while attaining a load balance threshold supplied by the user (in the experiments an imbalance as high as 95% is allowed). A breadth-first search of the tree is performed to search for such a set of subtrees. The remaining nodes of the tree are partitioned in a round robin fashion. Improving upon the previous two algorithms, Pothen and Sun [175] propose the *proportional mapping* algorithm. They observe that the remaining nodes after the subtree mapping can be assigned to the processors in order to reduce the communication overhead that arises because of the round robin scheme. The essential idea is to map the nodes (they use a clique tree representation of the filled-in graph and therefore nodes are cliques of the nodes of the original graph) that lie on the paths from already assigned subtrees to the root onto the processors that are associated with those subtrees. A first-fit decreasing bin-packing heuristic is used to select a processor among the candidates. In a different framework, Gilbert and Schreiber [107] address the parallelization on a massively parallel, fine-grained architecture (using a virtual processor per each entry of L). In this work, a submatrix corresponding to supernodes is treated as a dense submatrix and is factorized in a square grid of processors. In order to facilitate parallelism among independent, dense submatrices, a two-dimensional bin-packing is performed. It is interesting to note the relevance of this work to current massively parallel architectures. Amestoy

et al. [8,9,10] generalize and improve the heuristics in [92,175] by taking memory scalability issues into account and by incorporating dynamic load balancing decisions for which some preprocessing is done in the analysis phase.

Many of the known results on out-of-core factorization methods are surveyed in a recent thesis by Agullo [1]. The thesis surveys some out-of-core solvers including [58,109,164,186,203], provides NP-completeness results for the problem of minimizing I/O volume in certain variations of factorization methods, and also develops polynomial time algorithms for some other variations. Reid and Scott discuss the design issues for `HSL_MA77`, a robust, state-of-the-art, out-of-core multifrontal solver for symmetric positive-definite systems [176] and for symmetric indefinite systems [177].

3.4 Block triangular form

Consider a permutation of a square, nonsingular sparse matrix that yields a block upper triangular form (BTF):

$$A = \begin{pmatrix} A_{11} & * & * & * \\ O & A_{22} & * & * \\ \vdots & \vdots & \ddots & * \\ O & O & \cdots & A_{pp} \end{pmatrix},$$

where each block on the diagonal is square and nonsingular and the nonzeros are confined to the block upper triangular part of the permuted matrix. If a permutation to this form is used when solving the linear system, the whole system can be solved as a sequence of subproblems, each involving a solution with one of the blocks on the diagonal.

The algorithms to obtain the BTF proceed in two steps, see e.g., [60,82] and [119]. First, a maximum cardinality matching on the bipartite graph representation is found, see [62,63]. In the case of a structurally full-rank matrix, this would be a perfect matching. Then, the matrix is nonsymmetrically permuted so that the matching entries are on the main diagonal. The directed graph of this matrix is then constructed, and its strongly connected components are found [200] which define the blocks on the diagonal. Efficient and very compact implementations in Fortran are provided in [71,72].

The block structure of the BTF is unique, apart from possible renumbering of the blocks or possible orderings within blocks, as shown in [61,80,81]. In other words, the same block structure would be obtained from any perfect matching. We note that any such matching contains nonzeros that are only in the diagonal blocks of the target BTF.

The BTF form is generalized to rectangular and unsymmetric, structurally rank deficient matrices by Pothen [171] and Pothen and Fan [173] following the work of Dulmage and Mendelsohn [82,80,81]. According to this generalization

any matrix has the following form

$$\begin{pmatrix} A_H & * & * \\ O & A_S & * \\ O & O & A_V \end{pmatrix},$$

where A_H is underdetermined (horizontal), A_S is square, and A_V is overdetermined (vertical). Each row of A_H is matched to a column in A_H, but there are unmatched columns in A_H; each row and column of A_S are matched; each column of A_V is matched to a row in A_V, but there are unmatched rows in A_V. Furthermore, Pothen and Fan [173] and Dulmage and Mendelsohn [82] give a finer structural characterization. The underdetermined matrix A_H can be permuted into block diagonal form, each block being underdetermined. The square block A_S can be permuted into upper BTF with square diagonal blocks, as discussed before. The overdetermined block A_V can be permuted into block diagonal form, with each block being overdetermined. Again, the fine permutation is unique [171], ignoring permutations within each fine block. The permutation to the generalized BTF is performed in three steps. In the first step, a maximum cardinality matching is found, not necessarily a perfect matching. Then each row that reaches an unmatched column through alternating paths (these rows are all matched, otherwise the matching is not of maximum cardinality) are put into the horizontal block, along with any column vertex in those paths. Then, a corresponding process is run to detect the columns and rows of the vertical block. Finally, the previous algorithm is run on the remaining full rank square block to detect its fine structure. Pothen [171] proves the essential uniqueness of the BTF for rectangular and structurally singular square matrices (see also [80,81]).

In recent work, we have presented a few observations on the BTF of symmetric matrices [78]. Firstly, the blocks A_H and A_V are transposes of each other. That is, the set of rows and the set of columns that define the horizontal block are equal to the set of columns and the set of rows that define the vertical block, respectively. Secondly, a fine block of the square submatrix A_S is such that either the set of its row indices is equal to the set of its column indices, or they are totally disjoint and there is another square block equal to the transpose of the block.

4 Iterative methods

A different but equally well known family of methods used for solving linear systems of the form

$$Ax = b \tag{3}$$

starts with an initial guess and, by successive approximations, obtains a solution with a desired accuracy. The computations proceed in iterations (hence the name *iterative methods*), where a set of linear vector operations and usually one or two sparse matrix-vector multiplication operations take place at each iteration. As one can easily guess, the presence of sparse matrices raises a number

of combinatorial problems. Indeed, there is a beautiful interaction between the parallelization of a certain class of iterative methods and combinatorial optimization which revolves around graph and hypergraph partitioning. As this is the subject of another survey [26], we do not cover this issue. Instead, we refer the reader to [41,57,125,205] for this interaction and to [79] for different aspects of parallelization (including a survey of earlier studies, the effect of granularity of parallelism, and algorithm design issues).

There are many aspects to iterative methods; here we restrict the discussion to some preconditioning techniques because of their combinatorial ingredients. *Preconditioning* refers to transforming the linear system (3) to another one which is easier to solve. A *preconditioner* is a matrix enabling such a transformation. Suppose that M is a nonsingular matrix which is a good approximation to A in the sense that M^{-1} well approximates A^{-1}, then the solution of the system $M^{-1}Ax = M^{-1}b$ may be much easier than the solution of the original system. There are alternative formulations as to how to apply the preconditioner: from the left, from the right, or both from left and right. For our purposes in this section, those formulations do not make any difference, and we refer the reader to a survey by Benzi [18] that thoroughly covers most of the developments up to 2002. We refer to some other surveys when necessary.

Among various preconditioning techniques, we briefly discuss incomplete factorization-based preconditioners, support graph preconditioners, and algebraic multigrids. We choose these preconditioners because of the strong presence of combinatorial issues and well-studied problems. We do not discuss the sparse approximate inverse preconditioning [113] neither their factored form [20], although the choice of pattern for the approximate inverse is a combinatorial issue (see [44] and the references therein), and sparse matrix ordering has a significant effect on the factored form of approximate inverse (see for example [22]). We also do not include a recent family of banded preconditioners [160] which raises various combinatorial problems that are currently being investigated.

4.1 Incomplete factorization-based preconditioners

As discussed in the previous section, fill-in usually occurs during the LU decomposition of a matrix A. By selectively dropping the entries computed during the decomposition, one can obtain an *incomplete LU factorization* (ILU) of A with a lower and an upper triangular matrix $\hat{L}$ and $\hat{U}$. The matrix $M = \hat{L}\hat{U}$, then can approximate the matrix A and hence can be used as a preconditioner. Benzi traces incomplete factorization methods back to the 1950s and 1960s by citing papers by Buleev and Varga, but credits Meijerink and van der Vorst [163] for recognizing the potential of incomplete factorization as a preconditioner for the conjugate gradient method.

As just mentioned, the essence of incomplete factorization is to drop entries in the course of the elimination process. Current methods either discard entries according to their position, value, or with a combination of both criteria.

Consider a pattern $\mathcal{S} \subseteq \{1,\ldots,n\} \times \{1,\ldots,n\}$ and perform the elimination process as in (1) but allow fill-in if the position is in $\mathcal{S}$ using the formulae

$$a_{ij}^{(k+1)} \leftarrow \begin{cases} a_{ij}^{(k)} - \left(a_{ik}^{(k)}/a_{kk}^{(k)}\right) a_{kj}^{(k)} , & \text{if } (i,j) \in \mathcal{S} \\ a_{ij}^{(k)}, & \text{otherwise} \end{cases} \tag{4}$$

for each major elimination step k ($k = 1,\ldots,n$) and $i,j > k$. Often, $\mathcal{S}$ is set to the nonzero pattern of A, in which case one obtains ILU(0), a *no-fill* ILU factorization. This was used in [163] within the context of incomplete Cholesky factorization.

A generalization was formalized by Gustafsson [118] (see in [15, p.259]). Gustafsson develops the notion of level of fill-in and drops fill-in entries according to this criterion. The initial level of fill-in for a_{ij} is defined as

$$lev_{ij} \leftarrow \begin{cases} 0 & \text{if } a_{ij} \neq 0 \text{ or } i = j, \\ \infty & \text{otherwise} , \end{cases}$$

and for each major elimination step k ($k = 1,\ldots,n$) during the elimination (4), the level of fill-in is updated using the formula

$$lev_{ij} = \min\{lev_{ij}, lev_{ik} + lev_{kj} + 1\} .$$

Given an initial choice of drop level ℓ, ILU(ℓ) drops entries whose level is larger than ℓ. Observe that the level of fill-in is a static value that can be computed by following the elimination process on graphs.

There have been many improvements upon the basic two incomplete factorization methods discussed above, resulting in almost always better preconditioners. However, these two methods are still quite useful (and effective for certain problems) because of the structural properties of the computed factors, as we shall see later when discussing the parallel computation of incomplete factorization preconditioners.

For example, Saad [188] develops a dual threshold strategy ILUT(τ, p), where a fill-in entry is dropped if its value is smaller than τ, and at most p fill-ins per row are allowed. For more on the variations and the properties of incomplete factorization-based preconditioners, we refer the reader to [18,165,190].

Orderings and their effects As for their complete factorization counterparts, incomplete factorization preconditioners are sensitive to the ordering of the elimination. Recall from Section 3 that, for a complete factorization, the ordering affects both the fill-in and stability of the factorization. For an incomplete factorization, in addition to these two effects, the ordering of the eliminations also affects the convergence of the iterative method. This last issue, although demonstrated by many, has yet to be understood in a fully satisfactory way. Benzi [18] cites 23 papers between the years 1989 and 2002 that experimentally investigate the effects of ordering on incomplete factorization preconditioners.

The first comparative study of the effect of ordering on incomplete Cholesky factorization was performed by Duff and Meurant [70]. The paper shows that, contrary to what was conjectured in [195], the number of iterations of the conjugate gradient method is not related to the number of fill-ins discarded but is almost directly related to the norm of the residual matrix $R = A - \bar{L}\bar{L}^T$. Chow and Saad [46] show that for more general problems the norm of the preconditioned residual $(\bar{L}\bar{U})^{-1}R$ is also important.

The general consensus of the experimental papers, starting with [70], including [21], strongly favour the use of RCM. Bridson and Tang [33] prove a structural result (using only the connectivity information on the graph of A) as to why RCM yields successful orderings for incomplete factorization preconditioning. One of the results showing why RCM works for IC(0) is based on $(\bar{L}\bar{L}^T)^{-1}$ being fully dense if and only if each column of $\bar{L}$ has a nonzero below the diagonal. Any ordering yielding such a structure is called a reversed graph traversal in [33] and RCM is shown to yield such a structure. We note that for the complete factorization case such characterizations were used before; for example the irreducibility characterization of A in terms of the structure of L (see [208] and [65]). The other result of [33] is based on the intuition that if the structures of L^{-1} and $\bar{L}^{-1}$ coincide, then the incomplete factor returned by IC(0) could be a good approximation. It is then shown that reversing an ordering that can be found by a graph search procedure that visits, at each step, a node that is adjacent to the most recently visited node (allowing backtracking) will order A so that the above condition holds. RCM does not yield such an ordering in general, but a close variant always will.

As an alternative to using the ordering methods originally designed for complete factorization, orderings specially designed for incomplete factorization have also been developed. In [52] a *minimum discarded fill* ordering, MDF, is proposed. The algorithm is considered as the numerical analogue of the minimum deficiency ordering (scheme S3 of [202]), and it corresponds to ILU(ℓ). The basic idea is to eliminate the node with the minimum discarded fill at each stage of the incomplete elimination in an attempt to minimize the Frobenius norm of the matrix of discarded elements. The method has been developed in [53] yielding two variants, both of which still are fairly expensive. D'Azevedo et al. deserve the credit for giving the static characterization of the factors in ILU(ℓ) in terms of the graph of the original matrix. In [47], two sets of ordering methods which use the values of the matrix elements are proposed. The first one is a variation of RCM where ties are broken according to the numerical values of the matrix entries corresponding to edges between the vertex and the already ordered vertices. The second one is based on minimum spanning trees, where at each node the least heavy edge is chosen to produce an ordering of the nodes. These algorithms use heuristics based on a theorem [47, Theorem 1] (the proof refers to the results of [163,53]) relating the element l_{ij} to the entries along the path joining vertices i and j in the original graph of an M-matrix.

In recent studies, such as [69,19,68], nonsymmetric orderings that permute rows and columns differently are used to permute large entries to the diagonal

before computing an incomplete preconditioner. Other more recent work that uses similar ideas includes [27] where pivoting is performed during incomplete elimination; [147] where fill-reducing ordering methods are interleaved with the elimination; and [191] where weighted matching-like algorithms are applied to detect a diagonally dominant square submatrix, which is then approximately factorized. Its approximate Schur complement is then constructed, on which the algorithm is applied recursively.

Blocking methods for the complete factorization are adapted to the incomplete factorization as well. The aim here is to speed up the computations as for complete factorization and to have more effective preconditioners (in terms of their effect on the convergence rate). A significant issue is that in certain incomplete factorization methods, the structure of the incomplete factors are only revealed during the elimination process. Ng et al. [166] present a technique for the incomplete Cholesky factorization that starts with the supernodal structure of the complete factors. If standard dropping techniques are applied to individual columns, the pre-computed supernodal structure is usually lost. In order to retain the supernodal structure as much as possible, Ng et al. either drop the set of nonzeros of a row in the current set of columns (the supernode) or retain that set. In order to obtain sparser incomplete factors, they subdivide each supernode so that more rows can be dropped.

In [130] and [189] blocking operations are relaxed in such a way that the supernodes are not exact, but are allowed to incur some fill-in. In the first step, the set of exact supernodes are found. Then, in [189], a compressed matrix is created from the exact supernodes, and the cosine-similarity between nodes or supernodes are computed to allow some inexact supernodes. In [130], inexact amalgamations are performed between the parents and children in the assembly tree with a threshold measuring the inexactness of the supernodes.

Another set of blocking approaches are presented in [89,168], explicitly for preconditioning purposes. Here, a large number of small dense blocks are found and permuted to the diagonal. The initial intention of these methods was to obtain block diagonal preconditioners, but the resulting orderings are found to be useful for point incomplete factorizations as well, see [21]. The blocking methods are fast (in general run in $\mathcal{O}(n+\tau)$ time although the current version finds a maximum product matching with MC64 as a preprocessor) and are provided in the PABLO library (`http://www.math.temple.edu/~daffi/software/pablo/`).

Many of the current state-of-the-art variations of ILU methods are provided in ILUPACK [28]. Other efforts include PETSc [16], IFPACK [192], and ITSOL (`http://www-users.cs.umn.edu/~saad/software/ITSOL/index.html`).

Benzi et al. [21] ask the following questions; answers to which will shed light into the effect of orderings on incomplete factorization preconditioners: (i) why does the choice of the initial node and the ordering within level sets affect the performance of (reverse) Cuthill-McKee? (ii) why does ILU(0) with a minimum degree ordering not suffer from the instability that occurs when a natural ordering is used, for the model problem or similar ones?

Parallelization The parallel computation of ILU preconditioners is often implemented in two steps. Firstly, the matrix is partitioned into blocks to create a high level parallelism where the ILU of the interior blocks can be performed independently. Secondly, dependencies between blocks are identified and sequential bottlenecks reduced to increase the parallelism.

The basic algorithm for no-fill ILU can be found in [190, p.398]. A parallel algorithm for a threshold based ILU preconditioner is given in [138]. In this work, after the initial partitioning and ILUT elimination of interior nodes, an independent set of boundary nodes is found using Luby's algorithm [159]. After elimination of these nodes, which can be done in parallel, fill-in edges are determined and added between the remaining nodes. Another independent set is then found and eliminated. The process is continued until all nodes have been eliminated. Hysom and Pothen [134] develop a parallel algorithm for a level-based ILU. They order each subdomain locally, ordering the interface nodes of each domain after the interior nodes. Then, a graph of subdomains is constructed that represents interactions between the subdomains. If two subdomains intersect, ordering one before the other introduces a directed edge from the first domain to the second one. Considering these directions, Hysom and Pothen colour the vertices of the subdomain graph to reduce the length of directed paths in this graph. The colour classes can again be found using Luby's algorithm. Hysom and Pothen impose constraints on the fill-in that can be obtained from a pure ILU(ℓ) factorization. This helps improve the parallelization. Their paper presents an improvement to the scheme outlined above and provides a fill-in theorem for the incomplete factorization.

4.2 Support graph preconditioners

Combinatorial structures have been used to construct and analyse preconditioners. For example, Rose [182] defines the R-regular splitting (matrix splitting is a form of preconditioning, see [207] for splitting methods) of singular M-matrices. Starting from a given choice of diagonal blocks, Rose reorders the blocks so that the vertices in a cycle (guaranteed to exist) are ordered consecutively. This ordering guarantees the convergence of any given block regular splitting for singular M-matrices. This was not true without this simple combinatorial tinkering of the given choice of diagonal blocks.

A more recent combinatorial preconditioner and a set of tools used in designing and proving the effectiveness of the constructed preconditioner is based on work by Vaidya [206]. Although Vaidya's manuscript is not published, his main theorem and the associated preconditioners are given in the thesis of his student Joshi [137, Chapter 5]. In this work, preconditioners for symmetric, positive definite, diagonally dominant matrices are constructed using a maximum weighted spanning tree of the associated undirected graph (the edge weights are equal to the absolute values of the corresponding matrix entries). In other words, some off-diagonal entries of the given matrix are dropped to obtain the preconditioner. In Joshi's thesis there is a condition on which entries of A to drop: an edge can be dropped if one can associate a single path in the graph

of the preconditioner matrix such that all edges in this path have a weight at least as large as the weight of the dropped edge. A maximum weighted spanning tree satisfies this condition. Any matrix containing that spanning tree and some additional edges also satisfies this condition. Joshi demonstrates the development on two dimensional regular grids. First, he separates the boundary nodes from the internal ones by removing all the edges between the boundary nodes and the internal ones but keeping the edges between boundary nodes. Then, he constructs a spanning tree of the internal nodes, and finally joins the boundary to this tree with a single edge (one of those removed previously).

The proof that such structures give effective preconditioners uses two *graph embedding* notions (Joshi uses the term embedding just to mean the representation of a grid by a graph). For simplicity, consider two graphs H and G defined on the same set of vertices. The embedding of H into G is a set of paths of G, such that each edge in H is associated with a single path in G. The *congestion* of an edge of G is the sum of the weights of such paths that pass through that edge, and the *dilation* of an edge of H is the length of the associated path in G. The maximum congestion of an edge of G and the maximum dilation of an edge of H define, respectively, the congestion and the dilation of the embedding. Vaidya's main result as stated by Joshi says that the condition number of the preconditioned system is less than the product of the congestion and the dilation of the embedding. We note that these graph embedding notions are also known in parallel computing literature and are used in studying the interconnection networks (see for example [146, p.39]).

The basic support-tree preconditioners and the graph embedding tools used in bounding the condition number of the preconditioned system were extended and generalized by Miller and his students Gremban and Guattery [111,112,114]. The extensions by Miller and Gremban include projecting the matrix onto a larger space and building support trees using Steiner trees (a Steiner tree forms a spanning tree of a graph with possibly additional vertices and edges). Vertex separators that are used to partition the underlying graph are defined as the additional nodes. The leaves of the constructed support-tree correspond to the nodes of the underlying graph, and the internal nodes correspond to vertex separators. This form of preconditioner is demonstrated to be more amenable to parallelization than the original support-tree preconditioners [111, Section 3.4]. Reif [179] also develops Vaidya's preconditioners and reduces the bounds on the condition number of the preconditioned matrix by using a weighted decomposition, i.e., by partial embedding of the edges into multiple paths. Similar decompositions are used and defined more clearly by Guattery [114] based on Gremban's thesis. Guattery uses the embedding tools to analyse incomplete Cholesky factorization as well.

As seen from the references cited in the above paragraph, the earlier papers on support tree preconditioners are not published in widely accessible journals, except the rather theoretical paper by Reif [179]. The preconditioners, the related tools that are used to analyse them, and the potential of the theory as a means to analyse preconditioners are presented by Bern et al. [24]. This paper

collects many results and refines them, at the same time extending the techniques to analyse modified incomplete Cholesky factorization (the dropped entries are added to the diagonal entry, keeping the row sums of the original matrix and the preconditioner matrix the same). The main tool that is used in bounding the condition numbers of the preconditioned matrix is called the splitting lemma. Suppose we use a preconditioner B for the matrix A, where A is symmetric and diagonally dominant with nonnegative off-diagonal entries, and B is symmetric positive semidefinite, and both A and B have zero row sums. One way to bound the maximum eigenvalue of $B^{-1}A$ is to split A and B into m parts as

$$A = A_1 + \cdots + A_m \quad \text{and} \quad B = B_1 + \cdots + B_m \quad ,$$

where each A_k and B_k are symmetric positive semidefinite. Proving $\tau B_k - A_k$ is positive semidefinite for all k gives a bound on $\lambda_{max}(B^{-1}A)$. A similar technique is used to bound $\lambda_{min}(B^{-1}A)$ so that the condition number of the preconditioned system given by $\lambda_{max}(B^{-1}A)/\lambda_{min}(B^{-1}A)$ can be bounded. The relation with the graph embedding concept is that each A_k represents an edge in the graph of A and each B_k represents the associated path in the graph of B. Let A_k and B_k correspond to the edge (i, j), i.e., to the nonzero $a_{ij} = a_{ji}$. The matrix A_k contains the weight of the edge a_{ij} in its entirety, whereas the matrix B_k contains fractions of the weights of the edges along the path associated with the corresponding edge so that the sum of the weights of the same edge in different paths add up to the weight of the edge in B. For example in Vaidya's construction, if b_{ij} represents the weight of an edge b in the graph of B, and if the edge b appears in paths associated with some a_{ij}, then each such a_{ij} contributes a_{ij} divided by the congestion of b to b_{ij}. The diagonal entries of A_k and B_k are set in such a way that the row sums are zero. The congestion due to the edge a_{ij} represented by A_k in the path represented by B_k is $|a_{ij}|/b_k$, where b_k is the minimum magnitude of a nonzero off-diagonal entry in B_k. The dilation d_k is the length of the associated path, hence one less than the number of non-null rows or columns of B_k. These two numbers can be used to show that $d_k|a_{ij}|/b_k B_k - A_k$ is positive semidefinite. The bound on λ_{max} is therefore $\max_{ij} d_k(|a_{ij}|/b_k)$.

Bern et al. [24] use the above tools to analyse Vaidya's maximum-weight spanning tree preconditioners. In this analysis, B's underlying graph is a maximum-weight spanning tree T. Loose asymptotic bounds for congestion and dilation can be computed as follows. Suppose there are m edges in the graph of A, then B will be split using m paths, each defined uniquely in T. If one allocates $1/m$ of the weight of each edge of T to each path, then the maximum congestion due to an edge a_{ij} in the associated path would be $\frac{a_{ij}}{a_{ij}/m} = m$. Furthermore, the dilation can be at most $n-1$, one less than the number of vertices. Hence, $\mathcal{O}(mn)$ is a loose upper bound on the maximum eigenvalue λ_{max} of the preconditioned system. The analysis for λ_{min} is easier: λ_{min} is at least 1, as each edge of B is already in A. Therefore, the preconditioned system has a condition number bound of $\mathcal{O}(mn)$. Another class of preconditioners that are based on maximum-weight spanning trees is also proposed by Vaidya and analysed in [24]. These preconditioners are built by partitioning the vertices of a maximum-weight spanning

tree T into t connected parts, and then enriching the edge set of T by adding the maximum weighted edge between any pair of parts (if two parts are already connected by an edge of T, nothing is done for these two). With this kind of preconditioners, the preconditioned system is shown to have a condition number of $\mathcal{O}(n^2/t^2)$ [24].

Later, Boman and Hendrickson [31] generalized these embedding tools to develop more widely applicable algebraic tools. Specifically, the tools can now address symmetric positive semidefinite matrices. The insights gained with this generalization are used to analyse the block Jacobi preconditioner and have enabled the development of new preconditioners [30]. Additionally, the support theory techniques have been extended to include all diagonally dominant matrices [31].

The work by Chen and Toledo [43] presents an easily accessible description of Vaidya's preconditioners and their implementation. They report mixed results with Vaidya's preconditioners: on certain problems (2D and 3D with diagonal dominance) remarkable performance is obtained but on some others (3D general) the performance is poorer than incomplete factorization-based standard preconditioners. Sophisticated algorithmic approaches which aim at constructing spanning trees yielding provably better condition numbers for the preconditioned matrix, and again provably better graph decompositions are given in [86,197,198]. The developments in these papers lead to nearly linear time algorithms for solving a certain class of linear systems. Another very recent study is by Koutis, again a student of Miller. Koutis [142] proposes preconditioners for Laplacian matrices of planar graphs. Koutis develops the preconditioners by aggregating graph-based preconditioners of very small subsystems. Furthermore, Steiner tree preconditioners [111] are extended and algebraic multigrid preconditioners are cast in terms of Steiner trees, yielding combinatorial implications for the algebraic preconditioners.

The research on support graph preconditioners is very active. It seems that much of the results are theoretical, focusing only on some particular classes of linear systems. Therefore, there is much to do in this relatively young intersection of combinatorics and linear system solution. For example, except for a few comments in [31], nothing is said for nonsymmetric matrices, not even for pattern symmetric ones. Although we have very little experience with support graph preconditioning methods, we think that they can help understand the effects of ordering methods for incomplete factorization preconditioners discussed in the previous subsection.

Support-graph preconditioners are available in the TAUCS library of iterative solvers `http://www.tau.ac.il/~stoledo/taucs/` and in PETSc [16].

4.3 Algebraic multigrid preconditioning

Algebraic multigrid preconditioners approximate a given matrix by a series of smaller matrices. Simply put, the system of equations is coarsened to a much smaller system, a system is solved and refined at each level to obtain a correction to the original solution. There are a number of combinatorial issues regarding

efficient parallelization of multigrid solvers, see Chow at el. [45]. Here, we will look at the coarsening operation which incorporates a number of combinatorial techniques. For a survey on algebraic multigrid, we refer the reader to [199].

In general, there are three coarsening approaches used in algebraic multigrid: classical coarsening (see, e.g., [187]), aggregation based coarsening (see, e.g., [204]), and graph matching (this is a relatively new method described in [141]).

In classical coarsening approaches of the type given in [187], the grid points are classified into coarse or fine points. The coarse points are used to define the coarser grid. In order to restrict the size of the coarser grid, such points are restricted to be a maximal independent set. As we have seen before, this can be achieved using Luby's algorithm [159]. Two modifications of Luby's algorithm are presented in [55] for the coarsening operation in the algebraic multigrid context. The modifications include directing Luby's algorithm to choose points that have a higher number of influenced points (that is, those that are connected to the chosen points by heavy weights) and removing certain points before running the algorithm.

In aggregation based coarsening [204], an aggregate is defined as a root point and its immediate neighbours for which a certain condition in the magnitudes of the coefficient between the neighbours and the root point is satisfied. A constraint on a root point is that the aggregate defined around it cannot be adjacent to another root point. Therefore, a maximal independent set in the square of the graph (a graph formed by adding edges between any two vertices that are connected by a path of length 2 in the original graph) of the fine grid is found to define the roots of the aggregates again using Luby's algorithm. The exposition suggests that Luby's algorithm is run on the square graph. The graph colouring heuristics, see e.g., [32,90], can be modified and used to reduce the space requirements by avoiding the construction of the square graph (similar applications of the distance-k graph colouring heuristics can also boost the performance of some other aspects of multigrid solvers [54] as well as the coarsening [4]).

In the matching based coarsening [141], the coarse grids are defined using simple graph matching heuristics. In this work, a matching is found on the graph of a fine grid, and the matched vertices are reduced to a single vertex in the coarser grid. The matching is of the cardinality matching type, but does not aim at maximizing the cardinality. An investigation in the paper shows that if the original matrix is an M-matrix, so are the coarser matrices.

Current state-of-the-art multigrid solvers include ML [91] and BoomerAMG [131]. PETSc [16] provide interfaces for a number of multigrid preconditioners. For a much larger list of multigrid solvers and related multilevel ones see `http://www.mgnet.org/mgnet-codes.html`.

5 Conclusions

In this review, we have been rather eclectic in our choice of topics to illustrate the symbiotic relationship between combinatorics and sparse linear algebra. This

is in part because other papers in this volume address specific subareas and in part because of our own interest and expertise. Although space and energy prevent us from going into significant detail, we have given a substantial number of references that should easily quench the thirst of anyone eager to dig more deeply.

We have discussed graph search algorithms in the spirit of depth- and breadth-first search methods; both weighted and unweighted bipartite matchings; spanning trees; and graph embedding concepts. We believe that these are the most important and useful tools of the trade, and hence by having some level of acquaintance with these concepts, a computational scientist will be able to start understanding many of the issues that arise in solving sparse linear systems and be able to see how combinatorial approaches can be used to solve them.

We hope that we have communicated to the reader that combinatorial optimization and graph theory play a dominant role in sparse linear system solution. This is a delightful combination as the discrete and continuous worlds often seem so far apart, yet the synergy created by the interaction of the two leads to developments and advances in both worlds. Much of the combinatorial material that we have discussed is fairly elementary and indeed most would be covered in the context of an undergraduate level discrete mathematics course, or a senior-level algorithms course. We view this very positively as it means that these basic techniques are accessible to many people. However, the way these elementary techniques are applied requires substantial conceptualization, both in casting a problem in combinatorial terms and in restructuring computational methods to accommodate the combinatorial results.

Acknowledgements

We thank our colleagues whose works enabled much of the issues covered in this paper. We also thank Patrick R. Amestoy and Jean-Yves L'Excellent for their contribution to the presented material. We also thank Ümit V. Çatalyürek for his comments on an earlier version of the paper.

References

1. E. Agullo, *On the out-of-core factorization of large sparse matrices*, PhD thesis, Ecole Normale Supérieure de Lyon, Lyon, France, 2008.
2. E. Agullo, A. Guermouche, and J.-Y. L'Excellent, *Reducing the I/O volume in an out-of-core sparse multifrontal solver*, in High Performance Computing – HiPC2007; 14th International Conference, S. Aluru, M. Parashar, R. Badrinath, and V. K. Prasanna, eds., vol. 4873 of Lecture Notes in Computer Science, 2007, pp. 260–280.
3. ———, *A parallel out-of-core multifrontal method: Storage of factors on disk and analysis of models for an out-of-core active memory*, Parallel Computing, 34 (2008), pp. 296–317.
4. D. M. Alber and L. N. Olson, *Parallel coarse-grid selection*, Numerical Linear Algebra with Applications, 14 (2007), pp. 611–643.

5. H. Alt, N. Blum, K. Mehlhorn, and M. Paul, *Computing a maximum cardinality matching in a bipartite graph in time* $\mathcal{O}(n^{1.5}\sqrt{m/\log n})$, Information Processing Letters, 37 (1991), pp. 237–240.
6. P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.
7. P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer methods in applied mechanics and engineering, 184 (2000), pp. 501–520.
8. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.
9. P. R. Amestoy, I. S. Duff, and C. Vömel, *Task scheduling in an asynchronous distributed memory multifrontal solver*, SIAM Journal on Matrix Analysis and Applications, 26 (2004), pp. 544–565.
10. P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing, 32 (2006), pp. 136–156.
11. C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1404–1411.
12. C. Ashcraft and R. Grimes, *The influence of relaxed supernode partitions on the multifrontal method*, ACM Transactions on Mathematical Software, 15 (1989), pp. 291–309.
13. C. Ashcraft and J. W. H. Liu, *A partition improvement algorithm for generalized nested dissection*, Tech. Report BCSTECH-94-020, Boeing Computer Services, Seattle, WA, USA, 1996.
14. ———, *Robust ordering of sparse matrices using multisection*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 816–832.
15. O. Axelsson, *Iterative solution methods*, Cambridge University Press, Cambridge, 1994.
16. S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Web page*. http://www.mcs.anl.gov/petsc, (last access: March, 2009), 2001.
17. S. Barnard and H. D. Simon, *A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems*, Concurrency: Practice and Experience, 6 (1994), pp. 101–117.
18. M. Benzi, *Preconditioning techniques for large linear systems: A survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.
19. M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1333–1353.
20. M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM Journal on Scientific Computing, 17 (1996), pp. 1135–1149.
21. M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorization preconditioning of nonsymmetric problems*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1652–1670.
22. M. Benzi and M. Tůma, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1851–1868.

23. C. Berge, *Two theorems in graph theory*, Proceedings of the National Academy of Sciences of the USA, 43 (1957), pp. 842–844.
24. M. Bern, J. R. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo, *Support-graph preconditioners*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 930–951.
25. M. V. Bhat, W. G. Habashi, J. W. H. Liu, V. N. Nguyen, and M. F. Peeters, *A note on nested dissection for rectangular grids*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 253–258.
26. R. H. Bisseling, *Combinatorial problems in high-performance computing.* Presentation at Dagstuhl Seminar on Combinatorial Scientific Computing (09061), February 2009.
27. M. Bollhöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Linear Algebra and its Applications, 338 (2001), pp. 201–218.
28. M. Bollhöfer, Y. Saad, and O. Schenk, *ILUPACK.* http://www-public.tu-bs.de/~bolle/ilupack/ (last access: March, 2009).
29. M. Bollhöfer and O. Schenk, *Combinatorial aspects in sparse elimination methods*, GAMM Mitteilungen, 29 (2006), pp. 342–367.
30. E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo, *Maximum-weight-basis preconditioners*, Numerical Linear Algebra with Applications, 11 (2004), pp. 695–721.
31. E. G. Boman and B. Hendrickson, *Support theory for preconditioning*, SIAM Journal on Matrix Analysis and Applications, 25 (2003), pp. 694–717.
32. D. Bozdağ, Ü. V. Çatalyürek, A. H. Gebremedhin, F. Manne, E. G. Boman, and F. Özgüner, *A parallel distance-2 graph coloring algorithm for distributed memory computers*, in Proceedings of 2005 International Conference on High Performance Computing and Communications (HPCC-05), L. T. Yang, O. F. Rana, B. Di Martino, and J. Dongarra, eds., vol. 3726 of Lecture Notes in Computer Science, 2005, pp. 796–806.
33. R. Bridson and W.-P. Tang, *A structural diagnosis of some IC orderings*, SIAM Journal on Scientific Computing, 22 (2001), pp. 1527–1532.
34. R. A. Brualdi, *The symbiotic relationship of combinatorics and matrix theory*, Linear Algebra and its Applications, 162–164 (1992), pp. 65–105.
35. ———, *Combinatorial Matrix Classes*, vol. 108 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, 2006.
36. R. A. Brualdi and D. M. Cvetković, *A Combinatorial Approach to Matrix Theory and its Applications*, Chapman & Hall/CRC Press, Boca Raton, 2009.
37. R. A. Brualdi and H. J. Ryser, *Combinatorial Matrix Theory*, vol. 39 of Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1991.
38. T. N. Bui and C. Jones, *A heuristic for reducing fill-in in sparse matrix factorization*, in 6th SIAM Conference on Parallel Processing for Scientific Computing, Norfolk, Virginia, USA, 1993, pp. 445–452.
39. R. Burkard, M. Dell'Amico, and S. Martello, *Assignment Problems*, SIAM, Philadelphia, PA, USA, 2009.
40. Ü. V. Çatalyürek and C. Aykanat, *A fine-grain hypergraph model for 2D decomposition of sparse matrices*, in Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS, San Francisco, CA, 2001.
41. Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 673–693.

42. U. V. Çatalyürek, C. Aykanat, and B. Uçar, *On two-dimensional sparse matrix partitioning: Models, methods, and a recipe*, SIAM Journal on Scientific Computing, 32 (2010), pp. 656–683.
43. D. Chen and S. Toledo, *Vaidya's preconditioners: Implementation and experimental study*, Electronic Transactions on Numerical Analysis, 16 (2003), pp. 30–49.
44. E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1804–1822.
45. E. Chow, R. D. Falgout, J. J. Hu, R. S. Tuminaro, and U. M. Yang, *A survey of parallelization techniques for multigrid solvers*, in Parallel Processing for Scientific Computing, M. A. Heroux, P. Raghavan, and H. D. Simon, eds., vol. 20 of Software, Environments, and Tools, SIAM, 2006, ch. 10, pp. 179–201.
46. E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, Journal of Computational and Applied Mathematics, 86 (1997), pp. 387–414.
47. S. S. Clift and W.-P. Tang, *Weighted graph based ordering techniques for preconditioned conjugate gradient methods*, BIT Numerical Mathematics, 35 (1995), pp. 30–47.
48. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, first ed., 1990.
49. E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 24th national conference, New York, NY, USA, 1969, ACM, pp. 157–172.
50. T. A. Davis, *Direct Methods for Sparse Linear Systems*, no. 2 in Fundamentals of Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006.
51. T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM Journal on Matrix Analysis and Applications, 18 (1997), pp. 140–158.
52. E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 944–961.
53. ——, *Towards a cost-effective ILU preconditioner with high level fill*, BIT Numerical Mathematics, 32 (1992), pp. 442–463.
54. H. de Sterck, R. D. Falgout, J. W. Nolting, and U. M. Yang, *Distance-two interpolation for parallel algebraic multigrid*, Numerical Linear Algebra with Applications, 15 (2008), pp. 115–139.
55. H. de Sterck, U. M. Yang, and J. J. Heys, *Reducing complexity in parallel algebraic multigrid preconditioners*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1019–1039.
56. J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 720–755.
57. K. D. Devine, E. G. Boman, and G. Karypis, *Partitioning and load balancing for emerging parallel applications and architectures*, in Frontiers of Scientific Computing, M. Heroux, A. Raghavan, and H. Simon, eds., SIAM, Philadelphia, 2006.
58. F. Dobrian, *External Memory Algorithms for Factoring Sparse Matrices*, PhD thesis, Old Dominion University, Norfolk, Virginia, USA, 2001.

59. J. J. Dongarra, J. Du Croz, I. S. Duff, and S. Hammarling, *A set of level 3 Basic Linear Algebra Subprograms.*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.
60. I. S. Duff, *Analysis of Sparse Systems*, PhD thesis, Oxford University, England, 1972.
61. ———, *On permutations to block triangular form*, Journal of the Institute of Mathematics and its Applications, 19 (1977), pp. 339–342.
62. ———, *Algorithm 575: Permutations for a zero-free diagonal*, ACM Transactions on Mathematical Software, 7 (1981), pp. 387–390.
63. ———, *On algorithms for obtaining a maximum transversal*, ACM Transactions on Mathematical Software, 7 (1981), pp. 315–330.
64. ———, *Full matrix techniques in sparse Gaussian elimination*, in Proceedings of 1981 Dundee Biennal Conference on Numerical Analysis, G. A. Watson, ed., vol. 912 of Lecture Notes in Mathematics, 1982, pp. 71–84.
65. I. S. Duff, A. M. Erisman, C. W. Gear, and J. K. Reid, *Sparsity structure and Gaussian elimination*, SIGNUM Newsletter, 23 (1988), pp. 2–8.
66. I. S. Duff, A. M. Erisman, and J. K. Reid, *On George's nested dissection method*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 686–695.
67. ———, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.
68. I. S. Duff and J. 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 (1999), pp. 889–901.
69. ———, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM Journal on Matrix Analysis and Applications, 22 (2001), pp. 973–996.
70. I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.
71. I. S. Duff and J. K. Reid, *Algorithm 529: Permutations to block triangular form*, ACM Transactions on Mathematical Software, 4 (1978), pp. 189–192.
72. ———, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Transactions on Mathematical Software, 4 (1978), pp. 137–147.
73. ———, *MA27–A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Tech. Report AERE R10533, HMSO, London, UK, 1982.
74. ———, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.
75. ———, *A note on the work involved in no-fill sparse matrix factorization*, IMA Journal on Numerical Analysis, 1 (1983), pp. 37–40.
76. ———, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 633–641.
77. I. S. Duff, D. Ruiz, and B. Uçar, *Computing a class of bipartite matchings in parallel.* Presentation at SIAM 13th Conference on Parallel Processing for Scientific Computing (PP08), Atlanta, GA, USA, March 2008.
78. I. S. Duff and B. Uçar, *On the block triangular form of symmetric matrices*, Tech. Report TR/PA/08/26, CERFACS, Toulouse, France, 2008.
79. I. S. Duff and H. A. van der Vorst, *Developments and trends in the parallel solution of linear systems*, Parallel Computing, 25 (1999), pp. 1931–1970.
80. A. L. Dulmage and N. S. Mendelsohn, *Coverings of bipartite graphs*, Canadian Journal of Mathematics, 10 (1958), pp. 517–534.
81. ———, *A structure theory of bipartite graphs of finite exterior dimension*, Trans. Roy. Soc. Can. Sec. III, 53 (1959), pp. 1–13.

82. ———, *Two algorithms for bipartite graphs*, SIAM Journal on Applied Mathematics, 11 (1963), pp. 183–194.
83. S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, *The Yale sparse matrix package I: The symmetric codes*, International Journal for Numerical Methods in Engineering, 18 (1982), pp. 1145–1151.
84. S. C. Eisenstat and J. W. H. Liu, *The theory of elimination trees for sparse unsymmetric matrices*, SIAM Journal on Matrix Analysis and Applications, 26 (2005), pp. 686–705.
85. ———, *A tree-based dataflow model for the unsymmetric multifrontal method*, Electronic Transactions on Numerical Analysis, 21 (2005), pp. 1–19.
86. M. Elkin, Y. Emek, D. A. Spielman, and S.-H. Teng, *Lower-stretch spanning trees*, SIAM Journal on Computing, 38 (2008), pp. 608–628.
87. C. M. Fiduccia and R. M. Mattheyses, *A linear-time heuristic for improving network partitions*, in DAC '82: Proceedings of the 19th Conference on Design Automation, Piscataway, NJ, USA, 1982, IEEE Press, pp. 175–181.
88. M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987), pp. 596–615.
89. D. Fritzsche, A. Frommer, and D. B. Szyld, *Extensions of certain graph-based algorithms for preconditioning*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2144–2161.
90. A. H. Gebremedhin, F. Manne, and A. Pothen, *What color is your Jacobian? Graph coloring for computing derivatives*, SIAM Review, 47 (2005), pp. 629–705.
91. M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, and M. G. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Report SAND2006-2649, Sandia National Laboratories, 2006.
92. G. A. Geist and E. G. Ng, *Task scheduling for parallel sparse Cholesky factorization*, International Journal of Parallel Programming, 18 (1989), pp. 291–314.
93. A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.
94. A. George and J. W. H. Liu, *An automatic nested dissection algorithm for irregular finite element problems*, SIAM Journal on Numerical Analysis, 15 (1978), pp. 1053–1069.
95. ———, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Transactions on Mathematical Software, 6 (1980), pp. 337–358.
96. ———, *A minimal storage implementation of the minimum degree algorithm*, SIAM Journal on Numerical Analysis, 17 (1980), pp. 282–299.
97. ———, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
98. ———, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.
99. A. George, J. W. H. Liu, and E. Ng, *Communication results for parallel sparse Cholesky factorization on a hypercube*, Parallel Computing, 10 (1989), pp. 287–298.
100. A. George and D. R. McIntyre, *On the application of the minimum degree algorithm to finite element systems*, SIAM Journal on Numerical Analysis, 15 (1978), pp. 90–112.
101. A. George, W. G. Poole, and R. G. Voigt, *Incomplete nested dissection for solving n by n grid problems*, SIAM Journal on Numerical Analysis, 15 (1978), pp. 662–673.
102. A. J. George, *Computer Implementation of the Finite Element Method*, PhD thesis, Stanford University, Stanford, CA, USA, 1971.

103. J. R. Gilbert, *A note on the NP-completeness of vertex elimination on directed graphs*, SIAM Journal on Algebraic and Discrete Methods, 1 (1980), pp. 292–294.
104. J. R. Gilbert and J. W. H. Liu, *Elimination structures for unsymmetric sparse LU factors*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 334–352.
105. J. R. Gilbert, C. Moler, and R. Schreiber, *Sparse matrices in MATLAB: Design and implementation*, SIAM Journal on Matrix Analysis and Applications, 13 (1992), pp. 333–356.
106. J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 862–874.
107. J. R. Gilbert and R. Schreiber, *Highly parallel sparse Cholesky factorization*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 1151–1172.
108. J. R. Gilbert and R. E. Tarjan, *The analysis of a nested dissection algorithm*, Numerische Mathematik, 50 (1987), pp. 377–404.
109. J. R. Gilbert and S. Toledo, *High-performance out-of-core sparse LU factorization*, in 9th SIAM Conference on Parallel Processing for Scientific Computing (CDROM), 1999, p. p.10.
110. R. Greenlaw, *A model classifying algorithms as inherently sequential with applications to graph searching*, Information and Computation, 97 (1992), pp. 133–149.
111. K. D. Gremban, *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*, PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1996.
112. K. D. Gremban, G. L. Miller, and M. Zagha, *Performance evaluation of a parallel preconditioner*, in 9th International Parallel Processing Symposium, Santa Barbara, April 1995, IEEE, pp. 65–69.
113. M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM Journal on Scientific Computing, 18 (1997), pp. 838–853.
114. S. Guattery, *Graph embedding techniques for bounding condition numbers of incomplete factor preconditioners*, Tech. Report ICASE Report No.97-47, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia, 1997.
115. A. Guermouche and J.-Y. L'Excellent, *Constructing memory-minimizing schedules for multifrontal methods*, ACM Transactions on Mathematical Software, 32 (2006), pp. 17–32.
116. A. Gupta, *Fast and effective algorithms for graph partitioning and sparse matrix ordering*, Tech. Report RC 20496 (90799), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY, USA, 1996.
117. ———, *Improved symbolic and numerical factorization algorithms for unsymmetric sparse matrices*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 529–552.
118. I. Gustafsson, *A class of first order factorization methods*, BIT Numerical Mathematics, 18 (1978), pp. 142–156.
119. F. G. Gustavson, *Finding the block lower-triangular form of a sparse matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York and London, 1976, pp. 275–289.
120. M. Halappanavar, *Algorithms for vertex-weighted matching in graphs*, PhD thesis, Old Dominion University, Norfolk, Virginia, USA, 2008.
121. M. Hall, Jr., *An algorithm for distinct representatives*, The American Mathematical Monthly, 63 (1956), pp. 716–717.

122. P. Hall, *On representatives of subsets*, Journal of the London Mathematical Society, s1-10 (1935), pp. 26–30.
123. M. T. Heath, E. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Review, 33 (1991), pp. 420–460.
124. P. Heggernes, S. C. Eisenstat, G. Kumfert, and A. Pothen, *The computational complexity of the minimum degree algorithm*, in Proceedings of NIK 2001—14th Norwegian Computer Science Conference, Tromsø, Norway, 2001, pp. 98–109.
125. B. Hendrickson and T. G. Kolda, *Graph partitioning models for parallel computing*, Parallel Computing, 26 (2000), pp. 1519–1534.
126. B. Hendrickson and R. Leland, *The Chaco user's guide, version 2.0*, Sandia National Laboratories, Alburquerque, NM, 87185, 1995.
127. ———, *A multilevel algorithm for partitioning graphs*, in Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, 1995, ACM, p. 28.
128. B. Hendrickson and A. Pothen, *Combinatorial scientific computing: The enabling power of discrete algorithms in computational science*, in High Performance Computing for Computational Science—VECPAR 2006, M. Dayde, M. L. M. Palma, L. G. A. Coutinho, E. Pacitti, and J. C. Lopes, eds., vol. 4395 of Lecture Notes in Computer Science, 2007, pp. 260–280.
129. B. Hendrickson and E. Rothberg, *Improving the run time and quality of nested dissection ordering*, SIAM Journal on Scientific Computing, 20 (1998), pp. 468–489.
130. P. Hénon, P. Ramet, and J. Roman, *On finding approximate supernodes for an efficient block-ILU(k) factorization*, Parallel Computing, 34 (2008), pp. 345–362.
131. V. E. Henson and U. M. Yang, *BoomerAMG: A parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2002), pp. 155–177.
132. A. J. Hoffman, M. S. Martin, and D. J. Rose, *Complexity bounds for regular finite difference and finite element grids*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 364–369.
133. J. E. Hopcroft and R. M. Karp, *An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs*, SIAM Journal on Computing, 2 (1973), pp. 225–231.
134. D. Hysom and A. Pothen, *A scalable parallel algorithm for incomplete factor preconditioning*, SIAM Journal on Scientific Computing, 22 (2001), pp. 2194–2215.
135. B. M. Irons, *A frontal solution program for finite-element analysis*, International Journal for Numerical Methods in Engineering, 2 (1970), pp. 5–32.
136. J. A. G. Jess and H. G. M. Kees, *A data structure for parallel L/U decomposition*, IEEE Transactions on Computers, 31 (1982), pp. 231–239.
137. A. Joshi, *Topics in Optimization and Sparse Linear Systems*, PhD thesis, Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, IL, USA, December 1996.
138. G. Karypis and V. Kumar, *Parallel threshold-based ILU factorization*, in Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM), New York, NY, USA, 1997, ACM, pp. 1–24.
139. ———, *MeTiS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 4.0*, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.
140. B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, The Bell System Technical Journal, 49 (1970), pp. 291–307.

141. H. Kim, J. Xu, and L. Zikatanov, *A multigrid method based on graph matching for convection-diffusion equations*, Numerical Linear Algebra with Applications, 10 (2003), pp. 181–195.
142. I. Koutis, *Combinatorial and algebraic tools for multigrid algorithms*, PhD thesis, Carnegie Mellon University, Pittsburgh, May 2007.
143. H. W. Kuhn, *The Hungarian method for the assignment problem*, Naval Research Logistics Quarterly, 2 (1955), pp. 83–97.
144. ———, *The Hungarian method for the assignment problem*, Naval Research Logistics, 52 (2005), pp. 7–21.
145. ———, *Statement for Naval Research Logistics*, Naval Research Logistics, 52 (2005), p. 6.
146. V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Desing and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., 1994.
147. I. Lee, P. Raghavan, and E. G. Ng, *Effective preconditioning through ordering interleaved with incomplete factorization*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 1069–1088.
148. R. J. Lipton, D. J. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM Journal on Numerical Analysis, 16 (1979), pp. 346–358.
149. R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM Journal on Applied Mathematics, 36 (1979), pp. 177–189.
150. J. W. H. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.
151. ———, *A compact row storage scheme for Cholesky factors using elimination trees*, ACM Transactions on Mathematical Software, 12 (1986), pp. 127–148.
152. ———, *On the storage requirement in the out-of-core multifrontal method for sparse factorization*, ACM Transactions on Mathematical Software, 12 (1986), pp. 249–264.
153. ———, *Equivalent sparse matrix reordering by elimination tree rotations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 424–444.
154. ———, *The minimum degree ordering with constraints*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 1136–1145.
155. ———, *The role of elimination trees in sparse factorization*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 134–172.
156. ———, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.
157. J. W. H. Liu, E. G. Ng, and B. W. Peyton, *On finding supernodes for sparse matrix computations*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 242–252.
158. W.-H. Liu and A. H. Sherman, *Comparative analysis of the Cuthill-McKee and the Reverse Cuthill–McKee ordering algorithms for sparse matrices*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 198–213.
159. M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM Journal on Computing, 15 (1986), pp. 1036–1053.
160. M. Manguoglu, A. Sameh, and O. Schenk, *PSPIKE: Parallel sparse linear system solver*, in In Proc. Euro-Par 2009 Parallel Processing, 2009, pp. 797–808.
161. F. Manne and R. H. Bisseling, *A parallel approximation algorithm for the weighted maximum matching problem*, in Parallel Processing and Applied Mathematics, R. Wyrzykowski, K. Karczewski, J. Dongarra, and J. Wasniewski, eds., vol. 4967 of Lecture Notes in Computer Science, 2008, pp. 708–717.

162. H. M. Markowitz, *The elimination form of the inverse and its application to linear programming*, Management Science, 3 (1957), pp. 255–269.
163. J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.
164. O. Meshar, D. Irony, and S. Toledo, *An out-of-core sparse symmetric-indefinite factorization method*, ACM Transactions on Mathematical Software, 32 (2006), pp. 445–471.
165. G. A. Meurant, *Computer Solution of Large Linear Systems*, vol. 28 of Studies in Mathematics and Its Applications, North-Holland, Amsterdam, Netherlands, 1999.
166. E. G. Ng, B. W. Peyton, and P. Raghavan, *A blocked incomplete Cholesky preconditioner for hierarchical-memory computers*, in Iterative methods in scientific computation IV, D. R. Kincaid and A. C. Elster, eds., vol. 5 of IMACS Series in Computational Applied Mathematics, IMACS, New Brunswick, NJ, USA, 1999, pp. 211–222.
167. M. Olschowka and A. Neumaier, *A new pivoting strategy for Gaussian elimination*, Linear Algebra and Its Applications, 240 (1996), pp. 131–151.
168. J. O'Neil and D. B. Szyld, *A block ordering method for sparse matrices*, SIAM Journal on Scientific and Statistical Computing, 11 (1990), pp. 811–823.
169. S. Parter, *The use of linear graphs in Gauss elimination*, SIAM Review, 3 (1961), pp. 119–130.
170. F. Pellegrini, *SCOTCH 5.1 User's Guide*, Laboratoire Bordelais de Recherche en Informatique (LaBRI), 2008.
171. A. Pothen, *Sparse Null Bases and Marriage Theorems*, PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1984.
172. ———, *Graph matchings in combinatorial scientific computing (vertex-weighted and parallel edge-weighted)*. Presentation at Dagstuhl Seminar on Combinatorial Scientific Computing (09061), February 2009.
173. A. Pothen and C.-J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Transactions on Mathematical Software, 16 (1990), pp. 303–324.
174. A. Pothen, H. D. Simon, and K.-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM Journal on Matrix Analysis and Applications, 11 (1990), pp. 430–452.
175. A. Pothen and C. Sun, *A mapping algorithm for parallel sparse Cholesky factorization*, SIAM Journal on Scientific Computing, 14 (1993), pp. 1253–1257.
176. J. K. Reid and J. A. Scott, *An out-of-core sparse Cholesky solver*, Tech. Report RAL-TR-2006-013, Computational Sciences and Engineering Department, Rutherford Appleton Laboratory, Oxon, OX11 0QX, England, 2006.
177. ———, *An efficient out-of-core sparse symmetric indefinite direct solver*, Tech. Report RAL-TR-2008-024, Computational Sciences and Engineering Department, Rutherford Appleton Laboratory, Oxon, OX11 0QX, England, December 2008.
178. J. H. Reif, *Depth-first search is inherently sequential*, Information Processing Letters, 20 (1985), pp. 229–234.
179. ———, *Efficient approximate solution of sparse linear systems*, Computers and Mathematics with Applications, 36 (1998), pp. 37–58.
180. J. Riedy and J. Demmel, *Parallel weighted bipartite matching and applications*. Presentation at SIAM 11th Conference on Parallel Processing for Scientific Computing (PP04), San Francisco, CA, USA, February 2004.

181. D. J. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Read, ed., Academic Press, 1972, pp. 183–217.
182. ——, *Convergent regular splittings for singular M-matrices*, SIAM Journal on Algebraic and Discrete Methods, 5 (1984), pp. 133–144.
183. D. J. Rose and R. E. Tarjan, *Algorithmic aspects of vertex elimination in directed graphs*, SIAM Journal on Applied Mathematics, 34 (1978), pp. 176–197.
184. D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM Journal on Computing, 5 (1976), pp. 266–283.
185. D. J. Rose and G. F. Whitten, *Automatic nested dissection*, in ACM 74: Proceedings of the 1974 annual conference, New York, NY, USA, 1974, ACM, pp. 82–88.
186. V. Rotkin and S. Toledo, *The design and implementation of a new out-of-core sparse Cholesky factorization method*, ACM Transactions on Mathematical Software, 30 (2004), pp. 19–46.
187. J. W. Ruge and K. Stüben, *Algebraic multigrid*, in Multigrid Methods, S. F. McCormick, ed., SIAM, Philadelphia, Pennsylvania, 1987, ch. 4, pp. 73–130.
188. Y. Saad, *ILUT: A dual threshold incomplete LU factorization*, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402.
189. ——, *Finding exact and approximate block structures for ILU preconditioning*, SIAM Journal on Scientific Computing, 24 (2003), pp. 1107–1123.
190. ——, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2nd ed., 2003.
191. ——, *Multilevel ILU with reorderings for diagonal dominance*, SIAM Journal on Scientific Computing, 27 (2005), pp. 1032–1057.
192. M. Sala and M. Heroux, *Robust algebraic preconditioners with IFPACK 3.0*, Tech. Report SAND-0662, Sandia National Laboratories, 2005.
193. R. Schreiber, *A new implementation of sparse Gaussian elimination*, ACM Transactions on Mathematical Software, 8 (1982), pp. 256–276.
194. J. Schulze, *Towards a tighter coupling of bottom-up and top-down sparse matrix ordering methods*, BIT Numerical Mathematics, 41 (2001), pp. 800–841.
195. H. D. Simon, *Incomplete LU preconditioners for conjugate-gradient-type iterative methods*, in Proceedings of the 1985 Reservoir Simulation Symposium, Dallas, February 1985, pp. 387–396.
196. B. Speelpenning, *The generalized element method*, Tech. Report UIUCDCS-R-78-946, Department of Computer Science, University of Illinois at Urbana-Champaign, Illinois, 1978.
197. D. A. Spielman and S.-H. Teng, *Solving sparse, symmetric, diagonally dominant linear systems in time* $\mathcal{O}(m^{1.31})$, in 44th Annual IEEE Symposium on Foundations of Computer Science, IEEE, 2003, pp. 416–427.
198. ——, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in STOC'04: Proceedings of the 36th annual ACM symposium on Theory of computing, New York, NY, USA, 2004, ACM, pp. 81–90.
199. K. Stüben, *A review of algebraic multigrid*, Journal of Computational and Applied Mathematics, 128 (2001), pp. 281–309.
200. R. E. Tarjan, *Depth-first search and linear graph algorithms*, SIAM Journal on Computing, 1 (1972), pp. 146–160.
201. ——, *Data Structures and Network Algorithms*, vol. 44 of CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia, PA, USA, 1983.

202. W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55 (1967), pp. 1801–1809.
203. S. Toledo and A. Uchitel, *A supernodal out-of-core sparse Gaussian-elimination method*, in 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007),, R. Wyrzykowski, K. Karczewski, J. Dongarra, and J. Wasniewski, eds., vol. 4967 of Lecture Notes in Computer Science, Springer-Verlag Berlin Heidelberg, 2008, pp. 728–737.
204. R. S. Tuminaro and C. Tong, *Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines*, in Supercomputing '00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), Washington, DC, USA, 2000, IEEE Computer Society, p. 5.
205. B. Uçar and C. Aykanat, *Partitioning sparse matrices for parallel preconditioned iterative methods*, SIAM Journal on Scientific Computing, 29 (2007), pp. 1683–1709.
206. P. M. Vaidya, *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners.* Unpublished manuscript presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991.
207. R. S. Varga, *Matrix Iterative Analysis*, Springer, Berlin, Heidelberg, New York, second ed., 2000.
208. R. A. Willoughby, *A characterization of matrix irreducibility*, in Numerische Methoden bei Graphentheoretischen und Kombinatorischen Problemen, L. Collatz, G. Meinardus, and H. Werner, eds., vol. 29 of International Series of Numerical Mathematics, Birkhäuser Verlag, 1975, pp. 131–143.
209. M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM Journal on Algebraic and Discrete Methods, 2 (1981), pp. 77–79.