UNSYMMETRIC ORDERING USING A CONSTRAINED MARKOWITZ SCHEME *

PATRICK R. AMESTOY †, XIAOYE S. LI ‡, AND STÉPHANE PRALET §

Abstract. We present a family of ordering algorithms that can be used as a preprocessing step prior to performing sparse **LU** factorization. The ordering algorithms simultaneously achieve the objectives of selecting numerically good pivots and preserving the sparsity. We describe the algorithmic properties and challenges in its implementation. By mixing the two objectives we show that we can reduce the amount of fill in the factors and reduce the number of numerical problems during factorization. On a set of large unsymmetric real problems, we obtained the median reductions of 12% in the factorization time, of 13% in the size of the **LU** factors, of 20% in the number of operations performed during the factorization phase, and of 11% in the memory needed by the multifrontal solver `MA41_UNS`. A by-product of this ordering strategy is an incomplete **LU**-factored matrix that can be used as a preconditioner in an iterative solver.

1. Introduction. Direct methods for sparse unsymmetric linear systems usually involves an analysis phase preceeding the effective **LU** factorization [2, 10, 14, 20, 21]. The analysis phase transforms $\mathbf{A}$ into $\tilde{\mathbf{A}}$ with better properties for sparse factorization. It exploits the structural information to reduce the amount of fill-in in the **LU** factors and exploits the numerical information to reduce the need for numerical pivoting during factorization.

Two separate steps can be used in sequence for these two objectives:

1. Scaling and maximum transversal algorithms are used to transform $\mathbf{A}$ into $\mathbf{A}_1$ with large entries in magnitude on the diagonal.
2. A symmetric fill-reducing ordering, which preserves the large diagonal, is used to permute $\mathbf{A}_1$ into $\tilde{\mathbf{A}}$ so that the factors of $\tilde{\mathbf{A}}$ are sparser than those of $\mathbf{A}_1$.

Thus, the ultimate factorization is

$$\mathbf{LU} = \mathbf{P}_3\mathbf{P}_2\mathbf{D}_r\mathbf{A}\mathbf{D}_c\mathbf{Q}_1\mathbf{P}_2^T\mathbf{Q}_3 \tag{1.1}$$

where $\mathbf{D}_r$ and $\mathbf{D}_c$ are diagonal scaling matrices, $\mathbf{Q}_1$ is a permutation obtained from the maximum transversal algorithm, $\mathbf{P}_2$ corresponds to the fill-reducing permutation, and $\mathbf{P}_3$ and $\mathbf{Q}_3$ are permutations corresponding to numerical pivoting during factorization.

It has been observed in [4] that permuting large entries on the diagonal (computing $\mathbf{Q}_1$ based on [17]) can significantly reduce the number of numerical problems during factorization. A standard way for finding $\mathbf{P}_2$ is to apply a symmetric ordering algorithm (e.g., `AMD` [1]) to the structure of $\mathbf{A}_1 + \mathbf{A}_1^T$, where $\mathbf{A}_1 = \mathbf{D}_r\mathbf{A}\mathbf{D}_c\mathbf{Q}_1$. A better algorithm, called *diagonal Markowitz with local symmetrization* (or `DMLS`), was developed in [5], which could exploit the unsymmetric structure of $\mathbf{A}_1$ and was shown to give sparser factors than with `AMD`.

* Part of this research was supported by a grant NSF-INRIA number NSF-INT-0003274. The work of the second author was supported in part by the Director, Office of Advanced Scientific Computing Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098, and was supported in part by the National Science Foundation Cooperative Agreement No. ACI-9619020, NSF Grant No. ACI-9813362. The work of the third author was partially supported by CERFACS, 42, av. G. Coriolis, 31057 Toulouse Cedex 01, France, during his PhD.

† Patrick.Amestoy@enseeiht.fr, ENSEEIHT-IRIT, 2, rue Camichel, BP 7122 - F 31071 Toulouse Cedex 7, France.

‡ xsli@lbl.gov, Lawrence Berkeley National Lab, MS 50F-1650, 1 Cyclotron Rd., Berkeley, CA 94720.

§ Stephane.Pralet@enseeiht.fr, ENSEEIHT-IRIT, 2, rue Camichel, BP 7122 - F 31071 Toulouse Cedex 7, France.

The above two-step approach has two drawbacks:

- The numerical treatment forces the fill-reducing ordering to restrict pivot selection on the diagonal of $\mathbf{A}_1$, and so to compute a symmetric permutation,
- The ordering phase does not have numerical information to select pivots.

To improve sparsity preservation and numerical quality of the preselected pivots, we describe in this paper a family of orderings that can select off-diagonal pivots using a combination of structural and numerical criteria. Based on a numerical preprocessing of the matrix we build a set of numerically acceptable pivots, referred to as matrix $\mathbf{C}$, that may contain off-diagonal entries. We then compute an unsymmetric ordering taking into account both the structure of $\mathbf{A}$ and the numerical information in $\mathbf{C}$. The $\mathbf{C}$ matrix serves as a **constraint matrix** for the pivot selection and non-trivial floating point operations can be performed on this matrix to update the characteristics of the pivots. The new algorithm is referred to as *constrained Markowitz with local symmetrization* (or `CMLS`).

In summary, this work extended and generalized the `DMLS` work in several ways:

1. We do not limit our choice of pivots to the maximum transversal of $\mathbf{D}_r\mathbf{A}\mathbf{D}_c$. Our pivots can be chosen from a constraint matrix $\mathbf{C}$ that includes a transversal but is not limited to this transversal.
2. The constraint matrix $\mathbf{C}$ is updated *both structurally and numerically* after each step of elimination. The final $\mathbf{C}$ is an incomplete $\mathbf{LU}$ factor of $\mathbf{D}_r\mathbf{A}\mathbf{D}_c$.

Thus, instead of computing the permutations $\mathbf{Q}_1$ and $\mathbf{P}_2$ of equation (1.1) in two separate steps, `CMLS` simultaneously computes row and column permutations $\mathbf{P}_2$ and $\mathbf{Q}_2$, and the final factorization is

$$\mathbf{LU} = \mathbf{P}_3\mathbf{P}_2\mathbf{D}_r\mathbf{A}\mathbf{D}_c\mathbf{Q}_2^T\mathbf{Q}_3. \tag{1.2}$$

We evaluated the new ordering algorithms using two state-of-the-art direct solvers: the multifrontal code `MA41_UNS` [2, 7] and the supernodal code `SuperLU_DIST` [27, 28]. In `MA41_UNS`, standard partial pivoting with a threshold value is applied to locally select numerically stable pivots within a so-called frontal matrix. It is possible that some variables cannot be locally eliminated and are postponed for later eliminations, which may result in an increase in the size of the $\mathbf{LU}$ factors and the number of operations compared with those predicted during analysis. In `SuperLU_DIST`, a static pivoting strategy is used and the pivotal sequence chosen during analysis is kept the same (i.e., $\mathbf{P}_3$ and $\mathbf{Q}_3$ are the identity matrices in equations (1.1) and (1.2)). Iterative refinement may be needed to improve the solution.

The rest of the paper is organized as follows. Section 2 introduces the main components of our algorithm. Section 3 defines the graph-theoretic notations and describe the use of local symmetrization in our context. Section 4 describes the algorithmic contributions of the proposed `CMLS` method. A full detailed presentation of our implementation is given in [32]. Section 5 analyses the results of the newly implemented `CMLS` algorithm when applied to real-life unsymmetric test cases.

2. Components of our unsymmetric ordering. Given a matrix $\mathbf{A}$, let $\mathcal{P}attern(\mathbf{A})$ be the set of nonzero entries of $\mathbf{A}$: $\mathcal{P}attern(\mathbf{A}) = \{(i,j)$ such that $a_{ij} \neq 0\}$. Our unsymmetric ordering consists of two main steps:

- **Step 1:** Based on a numerical pre-treatment of the matrix $\mathbf{A}$, we extract a set of numerically acceptable pivots, referred to as the **constraint matrix** $\mathbf{C}$. We have $\mathcal{P}attern(\mathbf{C}) \subseteq \mathcal{P}attern(\mathbf{A})$, and if $c_{ij} \neq 0$ then $c_{ij} = a_{ij}$.

- **Step 2 :** Constrained unsymmetric ordering: the constraint matrix is used at each step of the symbolic Gaussian elimination to control the set of eligible pivots (possibly with respect to both numerical and structural criteria).

Before describing these two steps more precisely, we introduce definitions and notations that will be used to describe our algorithms.

Let $\mathbf{M} = (m_{ij})$ be a matrix of order n. If $\mathbf{M}$ can be permuted to have n nonzeros on the diagonal then $\mathbf{M}$ is **structurally nonsingular**. Let $G_M = (V_r, V_c, E)$ be the bipartite graph associated with the matrix $\mathbf{M}$. V_r is the set of row vertices and V_c is the set of column vertices. Let $(i,j) \in V_r \times V_c$ then $(i,j) \in E$ if and only if $m_{ij} \neq 0$. A **matching** is a subset of edges $\mathcal{M} \subseteq E$ such that for all vertices $v \in V_r \cup V_c$, at most one edge of $\mathcal{M}$ is incident on v. If $\mathbf{M}$ is structurally nonsingular, then there exists a matching $\mathcal{M}$ with n edges and $\mathcal{M}$ is said to be a **perfect matching**. We will also say that $\mathcal{M}$ is a **perfect transversal**.

For the sake of clarity, in the remainder of this paper we assume that $\mathbf{A}$ is structurally nonsingular. The adaptation of our algorithms to structurally singular matrices is straightforward but would have severely complicated our notations and comments.

2.1. Step 1: Numerical preprocessing. The objective of this preprocessing step is to extract the most significant (structurally and numerically) entries of the matrix $\mathbf{A}$ and to use them to build the constraint matrix $\mathbf{C}$.

Firstly, we scale the matrix $\mathbf{A}$ with the diagonal matrices $\mathbf{D}_r$ and $\mathbf{D}_c$, resulting in $\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$. The objective of this scaling is to homogenize the magnitude of the entries of the matrix. In particular it helps us to compare the magnitude of entries belonging to different rows and columns and thus to decide which entries will be selected in our *constraint matrix* $\mathbf{C}$. In this paper we use the scaling computed with the maximum weighted matching algorithm [17]. All entries in our scaled matrix then have entries lower than 1 in magnitude with a perfect transversal with entries of magnitude equal to 1.

Secondly, a *constraint matrix* $\mathbf{C}$ can be constructed from $\mathbf{A}$ such that $\mathcal{P}attern(\mathbf{C}) \subseteq \mathcal{P}attern(\mathbf{A})$ and $\mathbf{C}$ satisfies certain numerical and/or structural properties. Since the entries in $\mathbf{C}$ correspond to the potential pivots for the subsequent step, we only keep a subset of bounded size (typically less than $3n$) of the largest entries in the scaled matrix. Furthermore, we want $\mathbf{C}$ to be structurally nonsingular and thus we add entries from $\mathbf{A}$ to guarantee that $\mathbf{C}$ includes a perfect transversal $\mathcal{M}$.

2.2. Step 2: Constrained unsymmetric ordering. Let $\mathbf{A}^1 = \mathbf{A}$ be the original matrix of order n and $\mathbf{A}^k$ be the reduced matrix after eliminating the first $k-1$ pivots (not necessarily on the diagonal). Let $\mathbf{C}^1 = \mathbf{C}$ be such that $\mathcal{P}attern(\mathbf{C}^1) \subseteq \mathcal{P}attern(\mathbf{A}^1)$. At each step k, a pivot p^k such that $p^k \in \mathcal{P}attern(\mathbf{C}^k)$ is selected. This selection may combine structural heuristics based on the structure of $\mathbf{A}^k$ (e.g., approximate Markowitz count, approximate minimum fill, etc.) and numerical heuristics carried by the $\mathbf{C}^k$ matrix. Matrix $\mathbf{A}^k$ is updated (remove the row and the column of the pivot and add fill-ins in the Schur complement). Matrix $\mathbf{C}^k$ is updated such that $\mathbf{C}^{k+1}$ remains structurally nonsingular and $\mathcal{P}attern(\mathbf{C}^{k+1})$ is included in $\mathcal{P}attern(\bar{\mathbf{C}^k})$, where $\bar{\mathbf{C}^k}$ is defined as the reduced matrix after the elimination of pivot p^k in $\mathbf{C}^k$. The structure of $\mathbf{A}^{k+1}$ contains the structure of $\bar{\mathbf{C}^k}$. This implies that $\mathcal{P}attern(\mathbf{C}^{k+1}) \subseteq \mathcal{P}attern(\mathbf{A}^{k+1})$. To keep $\mathbf{C}^k$ structurally nonsingular, a perfect matching in $\mathbf{C}^k$ is maintained at each step. When there is no ambiguity, we will omit the superscript k from the matrix notations.

The following two considerations influence the update that will be performed on **C**:

- Which metric do we use to select a pivot?
- Which entries and/or values are added/updated in **C** at each step of the elimination?

Note that if we consider the magnitude of **C**'s entries to select a pivot, both the pattern of **C** and the numerical values need be stored and updated. Furthermore, any structural information about each entry (i, j) in **C** should carry information on the reduced matrix associated with the complete matrix **A**.

The ordering algorithm also depends on how **C** is updated at each step. As mentioned before in the description of **Step 2**, we want at each step to guarantee that:

(2.1) **C** must remain structurally nonsingular,

$$\mathcal{P}attern(\mathbf{C}^{k+1}) \subseteq \mathcal{P}attern(\bar{\mathbf{C}}^{k}). \tag{2.2}$$

3. Notations and definitions. Before giving the algorithmic details of the proposed `CMLS` method, we introduce the graph structures and notations that will be used in this paper. We first describe the main properties of bipartite graphs and bipartite quotient graphs and their relationship with Gaussian elimination. We then introduce the notations that will be used to describe our algorithms and define local symmetrization [5], a technique that simplifies the bipartite quotient graph implementation. Note that we use calligraphic font for notations related to quotient graphs and Roman font for other graphs.

3.1. Bipartite graph. Let $\mathbf{M} = (m_{ij})$ be a matrix and $G_M = (V_r, V_c, E)$ be its associated bipartite graph. Let R_i denote the structure of row i, i.e., $R_i = \{j \in V_c \ s.t. \ (i,j) \in E\}$. Let C_j denote the structure of column j, i.e., $C_j = \{i \in V_r \ s.t. \ (i,j) \in E\}$.

In Gaussian elimination, when a pivot $p = (r_p, c_p)$ is eliminated, a new matrix, referred to as the reduced matrix $\bar{\mathbf{M}}$ is computed. $\bar{\mathbf{M}}$ is obtained from **M** by removing row r_p and column c_p and by adding the Schur complement entries. In terms of graph manipulations, this elimination adds edges in the bipartite graph of **M** to connect all the rows adjacent to c_p to all the columns adjacent to r_p. This set of connected rows and columns is referred to as a **bi-clique**.

The symbolic factorization of **M** is done by building $\mathbf{M}^k$ for $k = 1$ to n, with $\mathbf{M}^1 = \mathbf{M}$. After eliminating the k^{th} pivot, we compute $\mathbf{M}^{k+1} = \bar{\mathbf{M}}^k$.

3.2. Bipartite quotient graph. In the previous section we have shown that, to update the bipartite graph we must add, at each elimination step, entries to the Schur complement matrix which may be costly to update and to store. It has been shown that quotient graphs can be used to efficiently model the factorization of symmetric matrices [20, 24]. The main idea is to use a compact representation of the cliques associated with the eliminated vertices. This concept can be extended (see [31]) to model the **LU** factorization. In this case, a bipartite quotient graph can be used to represent the edges in a bi-clique. It has then been shown in [31] that doing so the elimination can be modeled in space bounded by the size of the original matrix **A**. In this section, we first explain why the quotient graph model leads to more complex algorithms on unsymmetric matrices than on symmetric matrices. We then briefly define element absorption and explain the use of local symmetrization to reduce

the quotient graph complexity. Finally we introduce notations that will be used to describe our algorithms.

Let $\mathcal{P}_r \subseteq 2^{V_r}$ and $\mathcal{P}_c \subseteq 2^{V_c}$ be two partitions of V_r and V_c respectively. We define the bipartite quotient graph $\mathcal{G}_A = (\mathcal{P}_r, \mathcal{P}_c, \xi)$ of $\mathbf{A}$ such that an edge $(\mathcal{I}, \mathcal{J})$ belongs to $\xi \subseteq \mathcal{P}_r \times \mathcal{P}_c$ if and only if there exists an edge in G between a node of $\mathcal{I}$ and a node of $\mathcal{J}$.

Let $\mathcal{G}_A^k$ be the bipartite quotient graph used to represent the structure of the reduced submatrix $\mathbf{A}^k$ after k steps of elimination. Initially the bipartite quotient graph $\mathcal{G}_A^1$ is initialized with the partitions $\mathcal{P}_r = \{\{i\}$ such that $i \in V_r\}$ and $\mathcal{P}_c = \{\{j\}$ such that $j \in V_c\}$. Thus it is equivalent to the bipartite graph G^1. At step k of Gaussian elimination, any eliminated pivot $e = (r_e, c_e)$ will be referred to as a **coupled row and column element**. All the row and column vertices that are not coupled elements are referred to as the row and column **variables** of $\mathcal{G}_A^k$. Both row and column vertices of the graph are thus partitioned into two sets composed of variables (uneliminated vertices) and **elements** (eliminated vertices). We then define $\mathcal{G}_A^k = (\mathcal{V}_r^k \cup \overline{\mathcal{V}}_r^k, \mathcal{V}_c^k \cup \overline{\mathcal{V}}_c^k, \mathcal{E}^k \cup \overline{\mathcal{E}}^k)$. When it is clear from the context, we will omit the superscript k. The vertices in $\mathcal{V}_r$ (respectively $\mathcal{V}_c$) correspond to the row (respectively column) variables. The vertices in $\overline{\mathcal{V}}_r$ (respectively $\overline{\mathcal{V}}_c$) correspond to the row (respectively column) elements. The edge set $\mathcal{E}$ is such that $\mathcal{E} \subseteq (\mathcal{V}_r \times \mathcal{V}_c)$ whereas $\overline{\mathcal{E}}$ is such that $\overline{\mathcal{E}} \subseteq (\mathcal{V}_r \times \overline{\mathcal{V}}_c) \cup (\overline{\mathcal{V}}_r \times \mathcal{V}_c) \cup (\overline{\mathcal{V}}_r \times \overline{\mathcal{V}}_c)$. With our definitions (i, j) is a nonzero entry in the reduced matrix at step k if and only if there exists a path joining i and j which only visits the elements and for which all the edges in the even positions correspond to already eliminated pivots. In other words, the structure of a row i at step k is the set of reachable columns j through all the paths of the form $i \rightarrow c_{e_1} \rightarrow r_{e_1} \ldots \rightarrow c_{e_l} \rightarrow r_{e_l} \rightarrow j$ where $e_t = (r_{e_t}, c_{e_t}), 1 \leq t \leq l$ are coupled elements. Similarly, the structure of a column j at step k is the set of reachable rows i through all the paths of the form $j \rightarrow r_{e_1} \rightarrow c_{e_1} \ldots \rightarrow r_{e_l} \rightarrow c_{e_l} \rightarrow i$. This process may involve paths of arbitrary length in $\mathcal{G}_A^k$ [31] and in particular through more than one coupled element. For example in Figure 3.1, we assume that the entry (r_p, c_{e_2}) is initially zero and corresponds to fill-in due to the elimination of element e_1. Because of the path $r_p \rightarrow c_{e_1} \rightarrow r_{e_1}$, we know that the row structure of r_p contains the row structure of e_1 and in particular the entry (r_p, c_{e_2}). We know also that the row structure of r_p contains the row structure of e_2 because of the path $r_p \rightarrow c_{e_1} \rightarrow r_{e_1} \rightarrow c_{e_2} \rightarrow r_{e_2}$.

In the context of sparse Cholesky factorization, an undirected quotient graph (the row and column vertices are merged) is preferred and commonly used to compute an ordering for symmetric matrices (e.g., Multiple Minimum Degree [29] and Approximate Minimum Degree [1]). The structure of the factors can be computed following the paths of length at most two in this quotient graph. There are no edges between the elements.

In the unsymmetric case, when a pivot $p = (r_p, c_p)$ is selected, if there exists a cycle of the form $r_p \rightarrow c_{e_1} \rightarrow r_{e_1} \ldots \rightarrow c_{e_l} \rightarrow r_{e_l} \rightarrow c_p \rightarrow r_p$ then, except for r_p and c_p, the row and column elements in the cycle are no longer needed to retrieve the structure of the remaining variables. This process will be called **element absorption** and is illustrated in Figure 3.1. This absorption can be explained by the two following remarks (see [31] for further details):

- the row and the column of p contain respectively the structures of the row elements and of the column elements in the cycle,

- if one of the elements is reachable from a variable i then the other elements in the cycle are also reachable from i (in particular p).

During the absorption, each path from i to an element in the cycle is thus replaced by an edge from i to the current pivot.

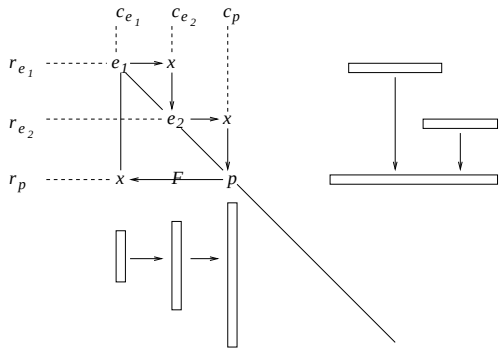


FIG. 3.1. *Illustration of a cycle* ($r_p \rightarrow c_{e_1} \rightarrow r_{e_1} \rightarrow c_{e_2} \rightarrow r_{e_2} \rightarrow c_p \rightarrow r_p$).

To avoid long search paths when we compute the structure of the row and the column of a pivot we decided to relax the element absorption rule as done in [5]. A row (resp. column) element is absorbed by the current row (resp. column) pivot if either it is adjacent to the column (resp. row) pivot or its associated column (resp. row) element is adjacent to the row (resp. column) pivot. This is referred to as **local symmetrization** in [5]. It implies that the resulting quotient graph $\mathcal{G}_A^k$ at step k models only an approximation of the structure of the reduced submatrix. It has been shown in [5] that the exploitation of element absorption combined with local symmetrization results in an in-place algorithm: at each step of the Gaussian elimination, the size of the quotient graph is bounded by the size of $\mathcal{G}_A^1$. Note that because of local symmetrization, an approximation of the symbolic factors can be computed following the paths of length at most three of the form $i \rightarrow c_e \rightarrow r_e \rightarrow j$ where (r_e, c_e) denotes a coupled row and column element. Note that applying local symmetrization is significantly different from symmetrization of the complete matrix. In fact we add at most $n-1$ virtual entries (at most one per absorbed element) and thus the structure of the factors computed with local symmetrization is equal to the real structure of the factors of a matrix $\mathbf{A}+\mathbf{D}$ where $\mathbf{D}$ has less that n entries.

To simplify the description of how the bipartite quotient graph is modified at each elimination step, we define $\overline{\mathcal{V}} \subseteq (\overline{\mathcal{V}}_r \times \overline{\mathcal{V}}_c)$ to be the set of coupled row and column elements corresponding to already eliminated pivots. Entries of the set $\overline{\mathcal{V}}$ will also be referred to as **coupled elements** or **elements** when it is clear from the context. Let $\mathcal{U}_p$ (resp. $\mathcal{L}_p$) be the column (resp. row) variables adjacent in $\mathcal{G}_A^k$ to the row (resp. column) element of a pivot $p = (r_p, c_p)$. Thanks to local symmetrization, the concept of absorption can be extended to coupled elements: an element $e = (r_e, c_e)$ such that $(r_p, c_e) \in \overline{\mathcal{E}}$ **or** $(r_e, c_p) \in \overline{\mathcal{E}}$ can be absorbed by p when p is selected as a pivot. A consequence of this absorption is that our ordering also generates a dependency graph between elements that is in fact a forest. This forest will be fully exploited by the unsymmetrized multifrontal approach [7].

For each row variable $i \in \mathcal{V}_r$ and column variable $j \in \mathcal{V}_c$, we define the element lists $\mathcal{R}_i$ and $\mathcal{C}_j$ as follows:

$$\mathcal{R}_i = \{e = (r_e, c_e) \in \overline{\mathcal{V}} \text{ s.t. } (i, c_e) \in \overline{\mathcal{E}}\}$$

and

$$\mathcal{C}_j = \{e = (r_e, c_e) \in \overline{\mathcal{V}} \text{ s.t. } (r_e, j) \in \overline{\mathcal{E}}\}.$$

Let $e = (r_e, c_e)$ be an element, if $e \in \mathcal{R}_i$ then we will say that **element e is adjacent to row variable** i. Similarly, if $e \in \mathcal{C}_j$ we will say that **element e is adjacent to column** j.

Using this notation, the adjacency of a row variable i (resp. column j) in $\mathcal{G}_A$ consists of a list of column variables denoted as $\mathcal{A}_{i*}$ (resp. a list of row variables $\mathcal{A}_{*j}$) and a list of elements $\mathcal{R}_i$ (resp. $\mathcal{C}_j$). Initially $\mathcal{R}_i = \mathcal{C}_j = \emptyset$ and $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$ corresponds to the original entries of $\mathbf{A}$. Each step of Gaussian elimination involves changes in the sets $\mathcal{R}_i$ and $\mathcal{C}_j$ as well as the computation of the structure of a current pivot p. The variable lists $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$ can also be pruned. Indeed, the edges in $\mathcal{G}_A$ between the variables and the elements implicitly represent the bi-clique of the element and can thus be used to remove the redundant entries in $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$. This important point will be further discussed in detail in Section 4.3.

When $(r_p, c_p) \in \mathcal{V}_r \times \mathcal{V}_c$ is selected as the next pivot we build the element p such that:

$$\mathcal{U}_p = \mathcal{A}_{r_p *} \cup \bigcup_{e \in \mathcal{R}_{r_p}} \mathcal{U}_e \cup \bigcup_{e \in \mathcal{C}_{c_p}} \mathcal{U}_e \tag{3.1}$$

and

$$\mathcal{L}_p = \mathcal{A}_{* r_p} \cup \bigcup_{e \in \mathcal{C}_{c_p}} \mathcal{L}_e \cup \bigcup_{e \in \mathcal{R}_{r_p}} \mathcal{L}_e. \tag{3.2}$$

The third term in each equation results from local symmetrization and will enable the current pivot to absorb all the elements which it was adjacent to. For example, let us assume that the entry $p1$ is selected as pivot in Figure 3.2. Since c_{p1} is adjacent to e_1, local symmetrization adds the virtual S_{p1} entry so that the row structure of $p1$ contains $\mathcal{U}_{e_1}$.

Let $\mathcal{F}_p = \mathcal{C}_{c_p} \cup \mathcal{R}_{r_p}$ be the set of elements adjacent to the current pivot. The elements in $\mathcal{F}_p$ are absorbed by p and the adjacency of each column variable j in $\mathcal{U}_p$ (resp. i in $\mathcal{L}_p$) is updated so that $\mathcal{C}_j \leftarrow (\mathcal{C}_j \setminus \mathcal{F}_p) \cup \{p\}$ (resp. $\mathcal{R}_i \leftarrow (\mathcal{R}_i \setminus \mathcal{F}_p) \cup \{p\}$). The structure of column j of the factors in the reduced matrix is then given by $\mathcal{A}_{*j} \cup \bigcup_{e \in \mathcal{C}_j} \mathcal{L}_e$. The structure of row i of the factors is $\mathcal{A}_{i*} \cup \bigcup_{e \in \mathcal{R}_i} \mathcal{U}_e$.

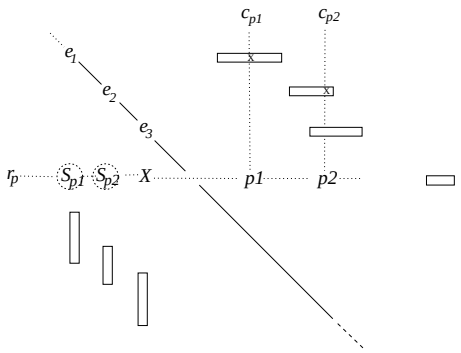


FIG. 3.2. *Influence of local symmetrization on the pivot structure*

Bipartite quotient graph of $\mathbf{A}$: $\mathcal{G}_A = (\mathcal{V}_r \cup \overline{\mathcal{V}}_r, \mathcal{V}_c \cup \overline{\mathcal{V}}_c, \mathcal{E} \cup \overline{\mathcal{E}})$	
$\mathcal{R}_i$	elements adjacent to row i
$\mathcal{A}_{i*}$	variables adjacent to row i
$\mathcal{C}_j$	elements adjacent to column j
$\mathcal{A}_{*j}$	variables adjacent to column j
$\mathcal{U}_p$	row structure of pivot p after its elimination.
$\mathcal{L}_p$	column structure of the pivot p after its elimination.
$\mathcal{F}_p$	elements that are adjacent to the row r_p or the column c_p of a non eliminated pivot p ($\mathcal{F} = \mathcal{R}_{r_p} \cup \mathcal{C}_{c_p}$)
$\mathcal{F}$	elements that are adjacent to row i or column j where i and j will be depend on the context ($\mathcal{F} = \mathcal{R}_i \cup \mathcal{C}_j$)

TABLE 4.1
Notations used for bipartite quotient graph.

Note that, although the above structural changes of the reduced submatrix are correct, they should not be used to estimate the structure of the factors. Indeed, if (i, j) were selected as the next pivot, then the correctly computed structure of the reduced matrix should include the local symmetrization terms (similar to equations (3.1) and (3.2)). In Figure 3.2, we illustrate the effect of local symmetrization on the structure of the selected pivot. Let us consider two candidate pivots belonging to the same row r_p, $p1 = (r_p, c_{p1})$ and $p2 = (r_p, c_{p2})$. We assume that all the elements in $\mathcal{G}_A$ adjacent to $p1$ and $p2$ are indicated in the figure. The structure of row r_p is then given by $\mathcal{A}_{r_p*} \cup \mathcal{U}_{e_3}$. This however does not give enough information on the structure of row r_p if either $p1$ or $p2$ were selected as the next pivot. If $p1$ were the next pivot then the structure of row r_p would be given by $\mathcal{U}_{p1} = \mathcal{A}_{r_p*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_1}$ because of the locally symmetrized entry S_{p1}. If $p2$ were the next pivot then the structure of row r_p would be given by $\mathcal{U}_{p2} = \mathcal{A}_{r_p*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_2}$ because of the locally symmetrized entry S_{p2}. This shows that, even if we cannot anticipate the effect of local symmetrization on the quotient graph $\mathcal{G}_A$ before the pivot selection, we should anticipate its effect on the metrics used to select the best pivot between $p1$ and $p2$.

4. CMLS algorithm. In this section, we describe the main features and properties of the `CMLS` algorithm. At each step of the algorithm, we need to know the exact structure of each row and column in $\mathbf{C}$. Moreover we need to compute a metric that reflects the quality of each nonzero entry in $\mathbf{C}$. It is thus natural to use a bipartite graph (with possibly weighted edges) for $\mathbf{C}$. Each edge corresponding to a nonzero entry may have one or more weights that will be used to select a pivot. For example, a numerical value that approximates the magnitude of the entries and a structural metric that approximates the Markowitz cost (i.e., the product of the row and column degree) can be used. On the other hand, in order to have a fast computation of a structural metric based on the pattern of $\mathbf{A}$ and to have an in-place algorithm, $\mathbf{A}$ is represented by its quotient graph and local symmetrization is employed. The notations used to represent the quotient graph at each step of the algorithm are summarized in Table 4.1.

In Section 4.1, we first describe the pivot selection algorithms. Updating the graph G_C and $\mathcal{G}_A$ associated with $\mathbf{C}$ and $\mathbf{A}$ respectively is discussed in Sections 4.2 and 4.3. In Section 4.4 we describe how to compute, at each step k and for each entry in the constraint matrix $\mathbf{C}^k$, structural metrics relative to $\mathcal{G}_A^k$. Section 4.5 finally explains how supervariables are defined and used in our context.

4.1. Pivot selection. At each step, the best pivot according to a given metric is selected. The metric choice determines the underlying algorithmic strategy. We say

that we use **structural strategies** in our algorithms when the entries are selected with respect to only information about the structure of the factors. In that case we will say that we use a **structural metric**. When we combine a **structural** metric and a **numerical** metric to select the pivot we will say that we use an **hybrid strategy**.

Moreover, in sparse matrix factorization, we also want to preserve the sparsity of the factors while controlling the numerical growth in the factors. Numerical thresholds are introduced to give freedom for the pivot selection to balance numerical precision with sparsity preservation. An entry $(i,j) \in \mathbf{C}^k$ is said to be **numerically acceptable** (or acceptable) according to a threshold τ if and only if $|c_{ij}| \geq \tau \times ||c_{.j}||_\infty$ where $\tau \in [0,1]$. To reduce the complexity of the algorithms, it is also common to limit the pivot search to a set of candidate pivots. For example in [13] the authors proposed to visit the entries of a fixed number of columns using the Zlatev-style search [37]. A similar strategy is used in [34] to find a pivot set in the context of parallel sparse **LU** factorization.

We use a slightly different algorithm: our pivot search is not restricted to columns but to a more complex set of entries in order to achieve a better fill-in reduction. At each step of the ordering, we look for the best entry $p = (r_p, c_p)$ within a subset (say S) of the entries in the bipartite graph G_C. The subset S is defined by two threshold parameters $\texttt{MS} > 0$ and $\texttt{ncol} \geq 0$ as follows. Firstly, the `MS` entries with the smallest structural metric m_0 are added to S. Secondly, those `MS` entries may belong to several columns. We then add in S all the other nonzero entries of those columns, but restricted to at most the first `ncol` columns. The set S is thus composed of a first set of `MS` entries, the so-called `MS`-set, and a second set, the so-called `ncol`-set $= S \backslash$ `MS`-set.

We now explain how we select the entry of minimum structural metric in S among the numerically acceptable pivots. We first visit the `MS`-set sorted in increasing order of the structural metric m_0. The first numerically acceptable entry found corresponds to the minimum with respect to our hybrid strategy and we stop the search. Otherwise, none of the values in the `MS`-set entries is numerically acceptable. However, if $\texttt{ncol} > 0$ then we are sure that at least `ncol` entries will be numerically acceptable since $\tau \leq 1$. Finally if $\texttt{ncol} = 0$ and none of the entries in the `MS`-set is numerically acceptable then the first entry of the `MS`-set is selected even if it is not numerically acceptable. (In our experiments $\texttt{MS} = 100$ and $\texttt{ncol} = 10$ are used.)

4.2. Update of the bipartite graph G_C. A bipartite graph is used to represent $\mathbf{C}$. At each step k, we need to add new entries in $G_{C^{k+1}}$ corresponding to the fill-ins in $\mathbf{C}^{k+1}$. Since G_C holds the set of candidate pivots, we need to guarantee that Properties (2.1) and (2.2) hold.

Let $\mathcal{M}$ be a matching in $\mathbf{C}^k$. The following two extreme strategies preserve these two properties:

- `MATCH_UPDATE` will refer to the strategy that performs incomplete Gaussian elimination on $\mathbf{C}$ to only preserve the perfect matching Property (2.1). Let $p = (r_p, c_p)$ be the current pivot. Let $(r_p, \texttt{match\_col})$ and $(\texttt{match\_row}, c_p)$ be the matched entries of $\mathbf{C}$ in row r_p and column c_p, respectively. That is, $(r_p, \texttt{match\_col}) \in \mathcal{M}$ and $(\texttt{match\_row}, c_p) \in \mathcal{M}$. If these entries are the same (i.e., (r_p, c_p) is a matched entry), nothing needs to be done to maintain Property (2.1). Otherwise entry $(\texttt{match\_row}, \texttt{match\_col})$ is added to $\mathbf{C}$ and $\mathcal{M}$ to maintain Property (2.1). Note that this entry corresponds to an entry in $\mathcal{P}attern(\bar{\mathbf{C}}^k)$, so that Property (2.2) remains true.
- `TOTAL_UPDATE` will refer to the strategy which performs all the updates in $\mathbf{C}$ (i.e., $\mathbf{C}^{k+1} = \bar{\mathbf{C}}^k$). Note that even if this strategy naturally preserves

Property (2.2), our perfect matching on $\mathbf{C}^{k+1}$ may have to be updated as in the MATCH_UPDATE strategy.

In practice, a mixed strategy, exploiting both MATCH_UPDATE and TOTAL_UPDATE will be used for the experiments. (The decision is based on memory and cost estimations of the algorithm.)

4.3. Update of the bipartite quotient graph $\mathcal{G}_A$. In Algorithm 4.1 we describe how the bipartite quotient graph associated with the reduced matrix is updated.

Algorithm 4.1 CMLS update of the bipartite quotient graph $\mathcal{G}_A^k$

Let $p = (r_p, c_p)$ be the current pivot at step k and $\mathcal{F}_p = \mathcal{R}_{r_p} \cup \mathcal{C}_{c_p}$.
if $\mathcal{U}_p \neq \emptyset$ and $\mathcal{L}_p \neq \emptyset$ **then**
  for each row $i \in \mathcal{L}_p$ **do**
1 $\mathcal{A}_{i*} = (\mathcal{A}_{i*} \setminus \mathcal{U}_p) \setminus \{c_p\}$ /* **variable elimination in row direction** */
2 $\mathcal{R}_i = (\mathcal{R}_i \setminus \mathcal{F}_p) \cup p$
  end for
  for each column $j \in \mathcal{U}_p$ **do**
3 $\mathcal{A}_{*j} = (\mathcal{A}_{*j} \setminus \mathcal{L}_p) \setminus \{r_p\}$ /* **variable elimination in column direction** */
4 $\mathcal{C}_j = (\mathcal{C}_j \setminus \mathcal{F}_p) \cup p$
  end for
else /* **pivot pruning** : delete all that is related to p, if $\mathcal{U}_p = \emptyset$ or $\mathcal{L}_p = \emptyset$ */
  for each row $i \in \mathcal{L}_p$ **do**
    $\mathcal{R}_i = (\mathcal{R}_i \setminus \mathcal{F}_p)$
    $\mathcal{A}_{i*} = \mathcal{A}_{i*} \setminus \{c_p\}$
  end for
  for each column $j \in \mathcal{U}_p$ **do**
    $\mathcal{C}_j = (\mathcal{C}_j \setminus \mathcal{F}_p)$
    $\mathcal{A}_{*j} = \mathcal{A}_{*j} \setminus \{r_p\}$
  end for
end if

The "if" block of the Algorithm 4.1 shows how the elements and variables are pruned. The element pruning performed at lines 2 and 4 includes pruning due to local symmetrization. The variable pruning performed at lines 1 and 3 removes the intersection of the adjacency structures. For each row i in $\mathcal{L}_p$, variables of $\mathcal{A}_{i*}$ that appear in $\mathcal{U}_p$ are removed and we say that we perform **variable elimination in the row direction**. For each column j in $\mathcal{U}_p$, variables of $\mathcal{A}_{*j}$ that appear in $\mathcal{L}_p$ are removed. This will be referred to as **variable elimination in the column direction**. We then say that our algorithm performs **variable elimination in one direction**. Note that if, at a given step, variables are removed from both row i and column i, it means that $i \in \mathcal{L}_p$ and $i \in \mathcal{U}_p$. In Section 4.3.1, we will prove that under additional assumptions more pruning of the variables could have been introduced. We then however comment in Section 4.3.2 that doing so makes impossible the detection of the reducibility as done in the "else" block of Algorithm 4.1. We will also explain why it is correlated with the strategy used to prune variables. Note that this additional pruning would have improved the accuracy of our structural metrics as explained in Section 4.4.

4.3.1. Two-way variable elimination. Property 4.1 shows that under additional assumptions the structure of the quotient graph can be further pruned.

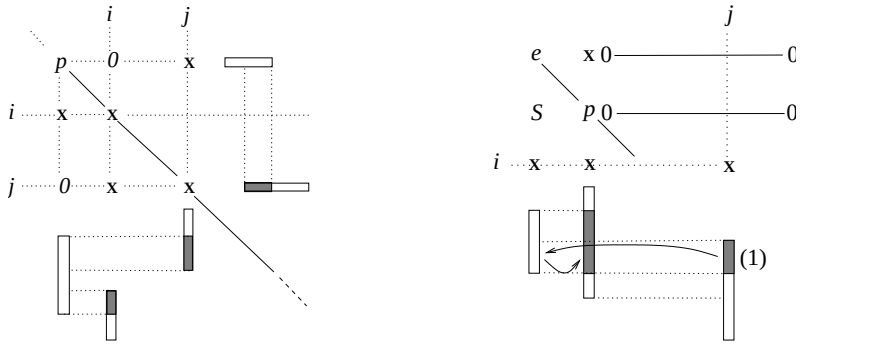
PROPERTY 4.1. *When local symmetrization is applied, and if at step k, the entry at position (i, j) is the only entry in row i and column j of $\mathbf{C}^k$ then:*

(1) if $i \in \mathcal{L}_p$, all the variables belonging to $\mathcal{L}_p$ can then be removed from $\mathcal{A}_{*j}$ (even though $j \notin \mathcal{U}_p$),

(2) if $j \in \mathcal{U}_p$, all the variables belonging to $\mathcal{U}_p$ can then be removed from $\mathcal{A}_{i*}$ (even though $i \notin \mathcal{L}_p$).

Proof. ¿From equation (2.2), we have $\mathcal{P}attern(\mathbf{C}^{k+1}) \subseteq \mathcal{P}attern(\bar{\mathbf{C}}^k)$. Therefore, if at step k, (i,j) is the only entry in row i and column j of $\mathbf{C}^k$, it will remain the only entry in its row and column for all subsequent $\mathbf{C}^l$, for $l > k$. Thus (i,j) will be selected as a pivot in a future step, and we can anticipate where local symmetrization will occur. So the entries in $\mathcal{A}_{*j} \cap \mathcal{L}_p$ for Property 4.1(1) (or in $\mathcal{A}_{i*} \cap \mathcal{U}_p$ for Property 4.1(2)) can be pruned and will be retrieved from $\mathcal{L}_p$ (or $\mathcal{U}_p$) when (i,j) is eliminated. □

When we apply Property 4.1, we say that the algorithm performs elimination in both row and column directions. This process will be referred to as **two-way variable elimination**. For example when the pivot choice is limited to a transversal, the two-way variable elimination can be performed at each step of the elimination, as in the `DMLS` algorithm [5]. This is illustrated in Figure 4.1(a). We assume, for the sake of clarity, that the input matrix has been permuted to have all the candidate pivots on the diagonal. The shaded areas correspond to the variables that can be removed from the variable adjacency lists because they are implicitly stored through the adjacency lists of element p.



(a) Illustration of two-way variable elimination.

(b) Effect of variable elimination in both directions on reducibility detection (S indicates the position of local symmetrization).

FIG. 4.1. *Variable elimination and reducibility detection.*

If the hypothesis of Property 4.1 is not true, the two-way variable elimination cannot be applied because we do not know whether local symmetrization will be performed or not. Let us consider Figure 4.1(a) again. If all three entries (i,i), (j,j) and (j,i) belong to $\mathbf{C}$, then we cannot prune all the shaded areas. This is because both (i,i) and (j,i) are potential pivots from column i. If (i,i) were chosen as the pivot from column i, then the shaded area in column i could have been pruned during the elimination of p thanks to local symmetrization relative to entry i in column $\mathcal{L}_p$. However, if (j,i) were selected as the pivot from column i, then since $j \notin \mathcal{L}_p$ and $i \notin \mathcal{U}_p$, the element p would not be used to build the row and column adjacency of (j,i). In this case the shaded area in column i should not be pruned during the elimination of p since it would be impossible to retrieve those variables. Note that the shaded area in column j can be pruned, because the entry (p,j) is *not a locally symmetrized entry*, and so the variable elimination in the column direction can be applied.

4.3.2. Reducibility detection. If the input matrix is reducible, we may encounter a pivot p such that either (1) both $\mathcal{L}_p = \emptyset$ and $\mathcal{U}_p = \emptyset$ (referred to as **strongly reducible**) or (2) $\mathcal{L}_p = \emptyset$ or $\mathcal{U}_p = \emptyset$ (referred to as **weakly reducible**). Ideally, we would like to remove p from the quotient graph $\mathcal{G}_A$ in both reducible cases. (In our context *removing a pivot* from $\mathcal{G}_A$ means that both the pivot and its adjacency structure can be suppressed without any further update of the graph.) However, we will show that whether p can be removed or not depends on whether we use only one-way variable elimination or use two-way variable elimination as well.

PROPERTY 4.2. *If the variable elimination is always done in one direction, then the current pivot p can be removed from the quotient graph if it is weakly reducible.*

Property 4.2 comes from the fact that the pruning of the structures due to local symmetrization has not been anticipated. Thus, none of the entries in $\mathcal{L}_p$ (if $\mathcal{U}_p = \emptyset$) or $\mathcal{U}_p$ (if $\mathcal{L}_p = \emptyset$) will be needed by the other variables to represent their adjacency structure in $\mathcal{G}_A$. Therefore, pivot p can be removed from $\mathcal{G}_A$.

PROPERTY 4.3. *If the two-way variable elimination has been done at least once, then the current pivot p can be safely removed from the quotient graph if and only if it is strongly reducible.*

Proof. Firstly, when $\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p = \emptyset$, p becomes a singleton element and can certainly be removed from the quotient graph. Secondly, let us suppose that two-way variable elimination has been performed at least once. We now build a counter example to show that we cannot safely (in all possible cases) remove a weakly reducible pivot p. Let us assume without loss of generality that $\mathcal{U}_p = \emptyset$ and $\mathcal{L}_p \neq \emptyset$. Let us assume that there exists a variable $i \in \mathcal{L}_p$ and that there is an entry (i, j) that is the only entry in row i and column j in $\mathbf{C}$. We also assume that variables in $\mathcal{A}_{*j}$ have been pruned under two-way variable elimination. Therefore, p must be used to retrieve those entries and cannot be removed from $\mathcal{G}_A$. This is illustrated in Figure 4.1(b) where the shaded area (1) in column j is first stored through element e then stored through element p (after pivot p absorbs element e). □

Property 4.3 indicates a drawback of the two-way variable elimination: we can only prune the pivot in the strongly reducible case. The algorithm may be very inefficient if the matrix is very reducible in the weak sense.

When the matrix is reducible to block triangular form (BTF), the reducibility detection may have a significant impact on the ordering quality [15]. In that case many instances in which an element is strongly or weakly reducible can appear. Property 4.2 can then be used to show that thanks to one-way variable elimination `CMLS` will better detect and exploit the BTF form of a matrix (see [6, 32] for further details).

4.4. Update of the structural metric. In this section, we describe heuristics to estimate the structural quality of a pivot.

In the preamble section, we first describe how we approximate the row and column degree. In Section 4.4.2, we describe a metric based on an upper bound on the fill-ins introduced at each step of elimination. This approximation of the fill-ins has been studied by the authors of `AMD` [1] for symmetric matrices. We provide a generalization of this approximation to unsymmetric matrices and prove that it is a tighter upper bound on the fill-ins than the approximations proposed for symmetric matrices in [33]. Note that concerning the deficiency approximation in [30], there is no guarantee that it is an upper bound of the fill-in. Our approximate minimum fill-in heuristic will be referred to as `AMFI`.

4.4.1. Preamble. Let us assume that the k^{th} pivot $p = (r_p, c_p)$ has been selected. All the entries in $(\mathcal{L}_p \times V_c \cup V_r \times \mathcal{U}_p) \cap \mathcal{P}attern(\mathbf{C})$ are involved in the

structural metric updates. The size of this area is thus larger than the area involved in the update of the structure of $\mathbf{C}$. The algorithm to update the structural metrics is one of the most costly steps of our algorithm.

We want the metrics to reflect the structural quality of an entry if it were selected as the next pivot. That is why we compute metrics which are related to the structure of our quotient graph and for which local symmetrization is anticipated. In the following, the degrees, approximate degrees, fill-ins and approximate fill-ins are all related to this quotient graph structure.

Let $d_r(i,j)$ and $d_c(i,j)$ denote respectively the external row and the external column degrees of entry (i,j). Similarly to `AMD` [1] and `DMLS` [5] algorithms, approximate row and column external degrees are computed. The `AMD`-like approximate external row and columns degree, $amd_r(i,j) > d_r(i,j)$ and $amd_c(i,j) > d_c(i,j)$ respectively, are then defined by the following two equations:

$$amd_r(i,j) = |\mathcal{A}_{i*} \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus j| + \sum_{e \in \mathcal{R}_i \cup \mathcal{C}_j} (|\mathcal{U}_e \setminus \mathcal{U}_p|) - \alpha_j, \quad \text{with } \alpha_j = \max(|\mathcal{C}_j|, 1) \text{ if } j \notin \mathcal{U}_p \text{ else } \alpha_j = 0. \tag{4.1}$$

$$amd_c(i,j) = |\mathcal{A}_{*j} \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus i| + \sum_{e \in \mathcal{C}_j \cup \mathcal{R}_i} (|\mathcal{L}_e \setminus \mathcal{L}_p|) - \beta_i, \quad \text{with } \beta_i = \max(|\mathcal{R}_i|, 1) \text{ if } i \notin \mathcal{L}_p \text{ else } \beta_i = 0. \tag{4.2}$$

As was done in [5], degree corrections (α_j and β_i in equations (4.1) and (4.2)) are introduced to improve the approximations of the row and column external degrees in the presence of local symmetrization. To justify these correction terms, one can observe that if $j \notin \mathcal{U}_p$ then j is counted in every $\mathcal{U}_e \setminus \mathcal{U}_p$ for e that is adjacent to column j ($e \in \mathcal{C}_j$). Furthermore, if $\mathcal{C}_j$ is empty and $j \notin \mathcal{U}_p$ then column j has been counted in $\mathcal{A}_{i*} \setminus \mathcal{U}_p$ and should then be subtracted. This explains the use of α_j in the correction. β_i can be justified in a similar way. The $|\mathcal{U}_e \setminus \mathcal{U}_p|$ and $|\mathcal{L}_e \setminus \mathcal{L}_p|$ quantities are computed similarly in the `AMD` algorithm.

Note that since only one-way variable elimination is employed, the computation of the metric is less accurate than with two-way variable elimination. This is because in the latter case, for any element e, row index $i \in \mathcal{U}_p$ and column index $j \in \mathcal{L}_p$, we have $\mathcal{A}_{i*} \cap \mathcal{U}_e = \emptyset$ and $\mathcal{A}_{*j} \cap \mathcal{L}_e = \emptyset$. This is no longer true when one-way variable elimination is used (see Algorithm 4.1). But as was explained in Section 4.3, the benefit of one-way variable elimination is to better exploit the BTF of the matrix.

After eliminating the k^{th} pivot, we approximate the row and column degree by

$$\overline{amd}_r(i,j) = \min(amd_r(i,j), n-k-1), \tag{4.3}$$

and

$$\overline{amd}_c(i,j) = \min(amd_c(i,j), n-k-1). \tag{4.4}$$

Note that these approximations do not use the values of the previous approximate row and column degrees because it would be costly to store these quantities for each entry in $\mathbf{C}$.

4.4.2. Approximation of the fill-in. We want to estimate the amount of new fill-in that would occur in the reduced matrix if an entry were selected as the next pivot. A coarse upper bound of the fill-in that would occur can be obtained by removing the area corresponding to $\mathcal{L}_p \times \mathcal{U}_p$ from the Markowitz cost or the area

corresponding to the largest adjacent clique [33]. A tighter approximation of the fill-in in the factors can be obtained by removing all the areas already filled during the elimination of the previous elements.

Suppose that $i \in \mathcal{L}_p$ or $j \in \mathcal{U}_p$. Let $\mathcal{F} = \mathcal{R}_i \cup \mathcal{C}_j$. Let e be an element that belongs to $\mathcal{F}$. Let $S(i,j)$ denote the union of the areas associated with all the elements adjacent to entry (i,j):

$$S(i,j) = \mid \bigcup_{e \in \mathcal{F}} (\mathcal{L}_e \setminus \{i\}) \times (\mathcal{U}_e \setminus \{j\}) \setminus (\mathcal{L}_p \times \mathcal{U}_p) \mid .$$

Ideally one might want to subtract both $\mid (\mathcal{L}_p \setminus \{i\}) \times (\mathcal{U}_p \setminus \{j\}) \mid$ and $S(i,j)$ from the Markowitz cost $d_r(i,j) \times d_c(i,j)$. An upper bound of the fill-in that would occur (including local symmetrization) if an entry (i,j) were eliminated is:

$$d_r(i,j)d_c(i,j) - \mid (\mathcal{U}_p \setminus \{j\}) \mid \mid (\mathcal{L}_p \setminus \{i\}) \mid - S(i,j).$$

The authors of [33] have observed that instead of using the exact external degrees one could use the approximate (in the sense of the `AMD` algorithm) external degrees since both produce results of comparable quality and since `AMD` based metrics are significantly faster to compute. In this context, the corresponding upper bound of the fill-in metric becomes

$$amd_r(i,j)amd_c(i,j) - \mid (\mathcal{U}_p \setminus \{j\}) \mid \mid (\mathcal{L}_p \setminus \{i\}) \mid - S(i,j). \tag{4.5}$$

Let AS be an overestimation of area S,

$$AS(i,j) = \sum_{e \in \mathcal{F}} \mid (\mathcal{L}_e \setminus \{i\}) \times (\mathcal{U}_e \setminus \{j\}) \setminus (\mathcal{L}_p \times \mathcal{U}_p) \mid . \tag{4.6}$$

Property 4.4 proves that one can in fact subtract area $AS(i,j)$, instead of $S(i,j)$ to obtain a more accurate upper bound of the fill metric than expression (4.5).

PROPERTY 4.4. *$amd_r(i,j)amd_c(i,j) - \mid (\mathcal{U}_p \setminus \{j\}) \mid \mid (\mathcal{L}_p \setminus \{i\}) \mid - AS(i,j)$ is an upper bound of the fill-in that would occur in the quotient graph if (i,j) were eliminated.*

An intuitive proof of Property 4.4 is that, during the computation of the approximate degree, the submatrix is expanded in such a way that the intersections between all $(\mathcal{U}_e \setminus \mathcal{U}_p \setminus \{j\})$ and between all $(\mathcal{L}_e \setminus \mathcal{L}_p \setminus \{i\})$ for $e \in \mathcal{F}$ are empty. The area AS corresponds to a real surface in the expanded matrix and can be removed from the area $amd_r(i,j)amd_c(i,j)$ to compute the fill-in that would occur in the expanded matrix. Moreover, this fill-in in the expanded matrix is an upper bound of the exact fill-in in the quotient graph. A formal proof of Property 4.4 is given in [32].

In practice we use $\overline{amd}_r(i,j)$ and $\overline{amd}_c(i,j)$ as defined in equations (4.3) and (4.4) instead of $amd_r(i,j)$ and $amd_c(i,j)$. Because of that it may happen that $\overline{amd}_r(i,j)\overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| - AS(i,j)$ becomes negative, meaning that either $\overline{amd}_r(i,j) < amd_r(i,j)$ or $\overline{amd}_c(i,j) < amd_c(i,j)$. In such cases, as it is done in `AMD` and `DMLS`, one can artificially set the metric to 0. We propose here an alternative, that could also be applied to these approaches to limit the tie-breaking. We introduce row and column scaling terms

$$\texttt{rowscale} = \frac{\overline{amd}_r(i,j) - |\mathcal{U}_p \setminus j|}{amd_r(i,j) - |\mathcal{U}_p \setminus j|} \text{ and } \texttt{colscale} = \frac{\overline{amd}_c(i,j) - |\mathcal{L}_p \setminus i|}{amd_c(i,j) - |\mathcal{L}_p \setminus i|}.$$

If one systematically scales the area AS by $\texttt{rowscale} \times \texttt{colscale}$, then we ensure a positive metric and avoid tie-breaking problems due to metrics equal to 0. Our final `AMFI` metric is then defined as follows:

$$(4.7) \quad metric^{(k+1)}(i,j) = \min \left\{ \begin{array}{ll} \overline{amd}_r(i,j)\overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| & \\ \quad - \texttt{rowscale} \times \texttt{colscale} \times AS(i,j) & \\ metric^{(k)}(i,j) \quad + |\mathcal{U}_p \setminus j| \times \overline{amd}_r(i,j) & \\ \quad + |\mathcal{L}_p \setminus i| \times \overline{amd}_c(i,j) & \\ \quad - 2 \times |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| & \end{array} \right.$$

4.5. Supervariables and mass elimination. For the sake of clarity, the algorithms described in the previous section did not include supervariables. In this section, we first define our generalization of **supervariables** and **mass elimination** to bipartite quotient graphs with off-diagonal pivots. We then revisit the previous algorithms and explain what has to be modified to detect and exploit supervariables.

In our context, we want supervariables to exploit identical adjacency structures in the graph at each step of the elimination. Supervariables are thus defined on the bipartite quotient graph of $\mathbf{A}$, whereas on the bipartite graph of $\mathbf{C}$ we only use simple variables. With the `CMLS` algorithm we cannot use exactly the same kinds of supervariables as in [1, 5, 19, 23] because they assume that pivots are on the diagonal so that a row can be associated with a column before being selected as pivot. That is why our concept of supervariable is closer to the one used in [22]: we define **indistinguishable row variables** (resp. **indistinguishable column variables**) as row variables (resp. columns variables) which have the same adjacency in $\mathcal{G}_A$. To limit the cost of supervariable detection, two hash functions (see for example [9]) are then used for each row and column direction.

If i and j are two indistinguishable row variables, they are replaced in $\mathcal{G}_A$ by a **row supervariable** containing both i and j, labeled by its **principal row variable** (i, say) [18, 19, 20]. The notation $\mathbf{i}$ is used to denote this row supervariable and $\mathbf{i} = \{i, j\}$. i and j are said to be **constituent row variables** of the row supervariable $\mathbf{i}$ and the notations $i \in \mathbf{i}$ and $j \in \mathbf{i}$ are then used. At the beginning of Gaussian elimination, the row variables are said to be **simple row variables**. Each simple row variable i can also be seen as a row supervariable $\mathbf{i} = \{i\}$. For each row supervariable $\mathbf{i}$, $|\mathbf{i}|$ corresponds to its size, i.e. its number of constituent variables. Similar definitions and notation can be introduced for the **column supervariables**, the **principal column variables**, the **constituent column variables** and the **simple column variables**. When it is clear from the context, we do not differentiate between a column or a row supervariable. Furthermore, let r_1 and r_2 be two row variables which belong to the same row supervariable $\mathbf{r}$ and c_1 and c_2 be two column variables which belong to the same column supervariable $\mathbf{c}$. After the elimination of pivot $p_1 = (r_1, c_1)$, $p_2 = (r_2, c_2)$ can be eliminated in $\mathcal{G}_A$ without causing extra fill-in. This process, commonly referred to as **mass elimination** [25], creates a new (super)element $e = (\mathbf{r}_e, \mathbf{c}_e)$ in the quotient graph with $\mathbf{r}_e = \{r_1, r_2\}$ and $\mathbf{c}_e = \{c_1, c_2\}$. In the following, we comment on the algorithmic modifications due to the introduction of supervariables.

Let p be the current pivot. The first modification of the algorithm concerns the introduction of a scaling of the structural metric as defined by equation (4.7). The structural metric of an entry $(\mathbf{i}, \mathbf{j})$ adjacent to p either in the row or column direction is divided by $min(|\mathbf{i}|, |\mathbf{j}|)$. Indeed $\min(|\mathbf{i}|, |\mathbf{j}|)$ corresponds to the size of the largest pivot block which could be eliminated if a pivot at the intersection of these row and column supervariables were selected.

The second modification of the algorithm concerns the elimination process which is performed in the following three main steps. During the first step, the scaled metric is used to select a pivot in $\mathbf{C}$. During the second step, we retrieve its associated row $\mathbf{r}_p$ and column $\mathbf{c}_p$ supervariable in $\mathbf{A}$. During the third step, we eliminate "as many as possible" variables belonging to $(\mathbf{r}_p \times \mathbf{c}_p) \cap \mathbf{C}$. Note that the meaning of "as many as possible" will depend on the context. If a hybrid strategy is used then pivot entries might be rejected because of numerical criteria. Furthermore, since the $\mathbf{C}$ matrix is updated when eliminating a pivot, the new nonzero entries that might be at the intersection of the pattern of $\mathbf{C}$ and the supervariables need also be considered. The same modified three steps are also applied when mass elimination process of supervariables adjacent to the current pivot is involved. Finally, if some constituent variables of a supervariable have not been eliminated, then they are used to build a new supervariable and are re-inserted in $\mathcal{G}_A$.

The final modification concerns the update of the structural metric. After the elimination of a pivot p, the approximate external row and column degrees as defined by equations (4.1) and (4.2) become:

$$amd_r(i,j) = |\mathcal{A}_{\mathbf{i}*} \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{\mathbf{j}\}| + \sum_{e \in \mathcal{R}_\mathbf{i} \cup \mathcal{C}_\mathbf{j}} (|\mathcal{U}_e \setminus \mathcal{U}_p|) - \alpha_\mathbf{j} \, |\mathbf{j}|, \quad \text{with } \alpha_\mathbf{j} = \max(|\mathcal{C}_\mathbf{j}|, 1) \text{ if } \mathbf{j} \notin \mathcal{U}_p \text{ else } \alpha_\mathbf{j} = 0. \tag{4.8}$$

$$amd_c(i,j) = |\mathcal{A}_{*\mathbf{j}} \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus \{\mathbf{i}\}| + \sum_{e \in \mathcal{R}_\mathbf{i} \cup \mathcal{C}_\mathbf{j}} (|\mathcal{L}_e \setminus \mathcal{L}_p|) - \beta_\mathbf{i} \, |\mathbf{i}|, \quad \text{with } \beta_\mathbf{i} = \max(|\mathcal{R}_\mathbf{i}|, 1) \text{ if } \mathbf{i} \notin \mathcal{L}_p \text{ else } \beta_\mathbf{i} = 0. \tag{4.9}$$

5. Experiments. In this section we analyze the effect of the `CMLS` ordering on the performance of sparse solvers. Our new ordering will be compared to the combination of `DMLS` ordering and `MC64` [16, 17] because it is the most robust in-place local heuristic (better than the combination of `AMD` and `MC64`, see [5]) in terms of numerical stability and fill-in reduction in the factors. `DMLS` takes into account the asymmetry of the matrices, selects pivots on the diagonal, applies local symmetrization and two-way variable elimination. Thus it can be considered a restricted `CMLS`. We recall that `MC64` permutes the matrix such that the product of the diagonal elements is maximized.

With the `CMLS` ordering, our pivot sequence results from a combination of structural and numerical information (even when only structural metrics are used to select the pivots, the initialization of our constraint matrix is based on numerical considerations). Therefore it is important to analyze the numerical quality of the proposed sequence of pivots. In this context, for very different motivations, we may want to experiment with both an approach that performs partial pivoting to preserve numerical stability and an approach based on static pivoting. In the first case, the numerical quality of the proposed sequence of pivots is not so critical to obtaining a backward stable factorization and we expect to improve the sparsity of the factors because of the freedom to select entries in the constraint matrix $\mathbf{C}$. In the case of a static pivoting, we expect that the capacity of `CMLS` to select pivots according to numerical criteria can be used to better control the numerical quality of the sequence while still offering more freedom than a diagonal Markowitz algorithm. In fact with the `CMLS` algorithm we can define a family of orderings and expect that two probably different members of this family can be used in these two cases: a `CMLS` ordering in which $\mathbf{C}$ offers a lot of freedom to choose the pivots and a `CMLS` ordering in which the selection of the pivots is strongly guided by the numerical values in $\mathbf{C}$.

To represent each class of solver techniques, we consider the multifrontal code MA41_UNS [2, 7] which performs numerical pivoting during the factorization and the supernodal code SuperLU_DIST [28] which performs static pivoting. Both codes are run in sequential mode. As shown in [4, 7, 12, 26] the approaches used to factorize the matrix in MA41_UNS and SuperLU_DIST are very competitive in shared/sequential and distributed memory environments respectively. Note that because of the important algorithmic similarities between MA41_UNS and the distributed memory code MUMPS [3], this work will be also very beneficial to the distributed memory multifrontal code.

In Section 5.1, we present our experimental environment. In Section 5.2, we discuss the case where the pivot choice in CMLS is restricted to the matching provided by MC64. In Section 5.3, we analyze the behavior of our ordering when a structural strategy is used to select the pivots. We report performance obtained with MA41_UNS in terms of time and memory used during factorization. In Section 5.4, we illustrate the benefits resulting from the use of hybrid strategies for pivot selection in SuperLU_DIST and focus on the numerical effects.

5.1. Experimental environment.

5.1.1. Test matrices and computing environment. Consider a matrix $\mathbf{A} = (a_{ij})$ and let $nnz(\mathbf{A})$ be its number of nonzero entries. We define the **structural symmetry** $s(\mathbf{A})$ as:

$$s(\mathbf{A}) = \frac{|\{(i,j) \text{ s.t. } a_{ij} \neq 0 \text{ and } a_{ji} \neq 0\}|}{nnz(\mathbf{A})}.$$

If $\mathbf{A}$ is symmetric,then $s(\mathbf{A}) = 1$, and if $\mathbf{A}$ is strictly triangular, then $s(\mathbf{A}) = 0$. In the remainder of this section, the symmetry of a matrix always refers to the structural symmetry after the MC64 permutation has been applied (see column sym of Table 5.1).

A representative set of 19 large unsymmetric matrices has been selected from Tim Davis' collection [11], see Table 5.1. Only matrices with a structural symmetry lower than 0.5 and of order greater than 10000 were chosen. Moreover, we limited the number of similar matrices from the same family to two in order to avoid the class effects. We also added to our test set four matrices (mixtank, invextr1, fidapm11 and cavity16) from the PARASOL test data[1] and Matrix Market[2], because we have observed that for these matrices SuperLU_DIST needs iterative refinement to improve the accuracy of the solution [4]. These four matrices will be used in Section 5.4 to illustrate that using the CMLS ordering improves the numerical behavior of SuperLU_DIST.

All our results have been obtained on a Linux PC computer (Pentium 4, 2.8 GHz, 2 GBytes of memory and 1 MByte of cache). We use the Portland Fortran 90 compiler pgf90, C compiler gcc (both with -O3 option) and ATLAS BLAS [35, 36].

We systematically apply random row and column permutations to our initial matrix so that the ordering algorithms are less sensitive to the effects of tie-breaking. We ran each problem with eleven random permutations and selected the run whose ordering returns the median fill-in in the factors. We did not observe large variations of the amount of fill-in in the factors from these random permutations except for matrix bbmat. (In general variations are smaller than 10%.)

[1] http://www.parallab.uib.no/projects/parasol/data
[2] http://math.nist.gov/MatrixMarket

Group/Matrix	n	nnz	sym	description
Vavasis/av41092	41092	1683902	0.08	Unstructured finite element
Hollinger/g7jac200sc	59310	837936	0.10	Economic model
Hollinger/g7jac180sc	53370	747276	0.10	Economic model
Hollinger/jan99jac120sc	41374	260202	0.16	Economic model
Hollinger/jan99jac100sc	34454	215862	0.16	Economic model
Mallya/lhr34c	35152	764014	0.19	Light hydrocarbon recovery
Mallya/lhr71c	70304	1528092	0.20	Light hydrocarbon recovery
Hollinger/mark3jac120sc	54929	342475	0.21	Economic model
Hollinger/mark3jac140sc	64089	399735	0.21	Economic model
Grund/bayer01	57735	277774	0.25	Chemical process simulation
Hohn/sinc18	16428	973826	0.27	Single-material crack problem (sinc-basis)
Hohn/sinc15	11532	568526	0.27	Single-material crack problem (sinc-basis)
Zhao/Zhao2	33861	166453	0.27	Electromagnetism
Sandia/mult_dcop_03	25187	193216	0.36	Circuit simulation
ATandT/twotone	120750	1224224	0.42	Harmonic balance method
ATandT/onetone1	36057	341088	0.42	Harmonic balance method
Norris/torso1	116158	8516500	0.43	Finite element matrices from bioengineering
Simon/bbmat	38744	1771722	0.49	2D airfoil, turbulence
Shen/shermanACb	18510	145149	0.50	Matrices from Kai Shen
mixtank	29957	1995041	0.91	fluid flow (PARASOL, Polyflow S.A.)
invextr1	30412	1793881	0.85	fluid flow (PARASOL, Polyflow S.A.)
fidapm11	22294	623554	0.45	CFD (SPARSKIT2 collection)
cavity16	4562	138187	0.84	Finite element modeling (SPARSKIT2 collection)

TABLE 5.1
Test matrices.

5.1.2. CMLS and DMLS testing environment. The initialization of $\mathbf{C}$ is done using a scaled matrix and the maximum weighted matching returned by `MC64`. To limit the size of $\mathbf{C}$ and the complexity (cost and memory) of the ordering phase, the initial number of entries in $\mathbf{C}^0$ is set between n and $4n$ (computation based on a function that depends on both n and $nnz(\mathbf{A})$). We then drop the entries that are smaller than 0.1 in magnitude and the entries whose structural metrics are too large. While dropping, we still maintain the nonsingularity property (2.1). In our test set, we observed that the size of $\mathbf{C}^0$ is between n and $3n$ after this last dropping phase.

We use the metric `AMFI` of Section 4.4.2 since it is the most efficient metric for both `CMLS` and `DMLS` orderings. In the `CMLS` implementation, we use `rowscale` and `colscale` coefficients (see end of Section 4.4.2) to reduce the amount of tie-breaking between variables that would have a negative metric (reset to 0) with `DMLS`. This algorithmic modification has also been implemented in the `DMLS` code to simplify our discussions in this section.

5.2. Preliminary remarks about diagonal constraint matrix. When $\mathbf{C}^0$ contains only the entries from the `MC64` matching and thus the set of candidate pivots for `CMLS` and `DMLS` is identical, one should expect a comparable behavior of the two algorithms in terms of fill-in in the factors. However, we have noticed that `CMLS` ordering tends to produce sparser factors (see [32] for detailed results) even if `DMLS` uses two-way variable elimination which leads to more accurate structural metrics, as explained in Section 4.4.1. This can be explained by the following algorithmic differences:

- Thanks to the one-way variable elimination, `CMLS` can eliminate all the elements in both the strongly reducible and the weakly reducible situations. This is well illustrated by the mult_dcop_3 matrix, which has 7448 irreducible components. `DMLS` and `CMLS` detect 875 singletons during a common

preprocessing step. Then during ordering, DMLS detects 95 additional blocks versus 229 blocks for CMLS.

- CMLS can create a row (column) supervariable if two rows (columns) have the same structure. DMLS can create a supervariable only if both rows and columns have the same structure. Thus, on the same quotient graph CMLS will detect more supervariables than DMLS. Note that the use of supervariables improves the accuracy of the structural metric. For example, if we consider that variables i and j belong to the same row supervariable, then the entries in $\mathcal{A}_{i*}$ and $\mathcal{A}_{j*}$ will not be counted as fill-in.

We should stress that these algorithmic differences were justified because CMLS is designed to handle more general and complex situations than DMLS. What was not at all predicted is that even a DMLS like algorithm—pivot choice limited to the diagonal—could benefit from the more general framework of the CMLS ordering.

5.3. Structural strategy.

5.3.1. Structure of the factors. In this section, we analyze the effect of the ordering on the size of the factors and compare the predicted size and the actual size of the factors. When there are no off-diagonal pivoting and node amalgamation, the actual size would be the same as the predicted size.

Table 5.2 compares CMLS with DMLS for both the estimated and the real size of the factors, using the MA41_UNS solver. For most matrices, the CMLS ordering results in sparser factors. The gains in sparsity vary from -22% to 56%, with gains very much comparable for both the estimated and the real size of the factors. On all matrices CMLS is either comparable or significantly better than DMLS except for lhr34c and lhr71c for which DMLS performs better. For the two matrices CMLS performs a lot of mass eliminations (approximatively 25% of the variables are eliminated during mass elimination). Instead of dividing our minimum fill-in estimation by the minimum of the sizes of the column and row supervariable, one could anticipate the number of mass eliminations and divide the AMFI estimation by the maximum of the sizes of the column and row supervariable. With this modification, we observed similar results in terms of fill-in between CMLS and DMLS for these two matrices.

As expected, the fact that more flexibility has been offered to select off-diagonal pivots in the constraint matrix helps CMLS to preserve the sparsity of the factors. However, in doing so we have allowed CMLS to select pivots that do not belong to the maximum weighted matching. Since a structural metric is then used by CMLS to select pivots, it is thus critical to evaluate the numerical quality of this pivot sequence with MA41_UNS. We recall that, thanks to partial threshold pivoting, the factorization phase of MA41_UNS (the default value of the threshold is used in all experiments) will modify the pivot sequence to control the growth of the size of the factors. This may result in an increase in the estimated factor size and number of operations. We thus also provide in Table 5.2 the ratio between the number of nonzeros in the factors and the forecast number of nonzeros in the factors. Note that from a software point of view it is also critical for the estimation to reflect reality. Clearly an accurate estimation is important for algorithms that are implemented without dynamic memory allocation. Even for C, C++ or FORTRAN90 based implementations that allow dynamic memory allocations, their cost may be not negligible. Finally, the accuracy of the memory estimation is even more critical in a distributed memory environment. For example, the buffers used for communications need to be well estimated. We see in Table 5.2 that the increase in the size of the factors is reasonable.

Matrix	Estimated size of factors				Real size of factors		Ratio: actual/predicted	
	CMLS	std	DMLS	std	CMLS	DMLS	CMLS	DMLS
av41092	6609	0.01	9323	0.02	6849	9553	1.03	1.02
g7jac200sc	27912	0.06	30424	0.02	28282	30443	1.01	1.00
g7jac180sc	24755	0.04	26789	0.03	25077	26810	1.01	1.00
jan99jac120sc	3191	0.02	4326	0.03	3197	4330	1.00	1.00
jan99jac100sc	2651	0.02	3373	0.03	2656	3376	1.00	1.00
lhr34c	4264	0.05	3571	0.03	4405	3668	1.03	1.02
lhr71c	9142	0.02	7189	0.02	9557	7377	1.04	1.02
mark3jac120sc	13333	0.02	12963	0.04	13386	12998	1.00	1.00
mark3jac140sc	15380	0.02	15093	0.01	15453	15136	1.00	1.00
bayer01	1253	0.03	2220	0.04	1253	2220	1.00	1.00
sinc18	26505	0.05	31722	0.04	27427	31926	1.03	1.00
sinc15	12596	0.03	15367	0.04	12917	15455	1.02	1.00
Zhao2	12258	0.01	14069	0.02	12588	14434	1.02	1.02
mult_dcop_03	713	0.01	940	0.07	714	906	1.00	0.96
twotone	7552	0.01	8458	0.02	7552	8458	1.00	1.00
onetone1	2913	0.02	3204	0.02	2921	3204	1.00	1.00
torso1	30656	0.02	34200	0.01	30656	34326	1.00	1.00
bbmat	38088	0.10	46436	0.14	38888	46471	1.02	1.00
shermanACb	362	0.01	426	0.01	362	426	1.00	1.00
Mean/Median	0.88/0.87				0.89/0.87			

TABLE 5.2

MA41_UNS *size of the factors and analysis reliability. Each number for the factor size is in thousands.* std*: standard deviation over the eleven runs. Mean (resp. Median) : mean (resp. median) value of the ratio* CMLS *statistic /* DMLS *statistic.*

5.3.2. Run-time and memory usage. In this section, we examine the number of operations, run-time and memory usage of MA41_UNS. The extra cost due to numerical pivoting during factorization is always included in the number of operations. Note that the timings for the factorization and the solution phase have to be interpreted carefully because they strongly depend on the basic linear algebra kernels used.

We see in Table 5.3 that on almost all the matrices, the CMLS ordering reduces the amount of memory used, with an average reduction around 11%. The reduction in the number of operations is even larger (median value of 20%) and will contribute to the reduction in the factorization time.

Table 5.4 then compares the time of the three main steps of the solution process. Note that the ordering time of both orderings depends on two opposite effects that are difficult to assess. The better we preserve sparsity, the smaller might be the quotient graph, and the faster we can process it. On the other hand, the better we preserve sparsity, the fewer the elements are absorbed, the fewer the supervariables are detected, and the higher the complexity might be. However, our new ordering is a real unsymmetric ordering that selects off-diagonal pivots and updates a constraint matrix. One should thus expect the time spent in the ordering to be higher with CMLS than with DMLS. Indeed, CMLS performs more metric computations and has to explicitly store and manipulate the constraint matrix $\mathbf{C}$. The metric update is the most costly step of the ordering so that the complexity of the ordering is tightly linked to the size of $\mathbf{C}$. Considering that the size of $\mathbf{C}^0$ is typically between $2n$ and $3n$, we see in Table 5.4 that CMLS is quite competitive with respect to DMLS (we observe that the cost of CMLS does not linearly increase with the size of $\mathbf{C}^0$). Two algorithmic differences might explain the good behaviour of the CMLS ordering (see for example torso1 matrix):

	Memory needed			Number of operations		
Matrix	CMLS	DMLS	ratio	CMLS	DMLS	ratio
av41092	7104	9998	0.71	1760	3533	0.49
g7jac200sc	29082	32912	0.88	27717	32553	0.85
g7jac180sc	26500	27566	0.96	24272	28190	0.86
jan99jac120sc	3245	4615	0.70	1042	1691	0.61
jan99jac100sc	2711	3579	0.69	867	1196	0.72
lhr34c	4501	3684	1.22	731	422	1.73
lhr71c	9786	7465	1.31	1949	852	2.28
mark3jac120sc	13909	13511	1.02	7524	6465	1.16
mark3jac140sc	15937	15963	1.00	8592	7558	1.13
bayer01	1256	2231	0.56	41	140	0.29
sinc18	32375	35409	0.91	49281	60152	0.81
sinc15	15031	16885	0.93	16515	19376	0.85
Zhao2	13200	15492	0.85	7622	9655	0.78
mult_dcop_03	746	969	0.76	51	117	0.43
twotone	8038	9006	0.89	4791	5412	0.88
onetone1	3437	3652	0.94	1035	1284	0.80
torso1	33759	36761	0.91	24620	36523	0.67
bbmat	39303	47156	0.83	31576	54061	0.58
shermanACb	394	452	0.87	20	30	0.66
Mean/Median			0.89/0.89			0.87/0.80

TABLE 5.3

MA41_UNS *memory used (in thousands of reals) and number of operations (in millions). ratio : ratio of* CMLS *statistic /* DMLS *statistic. Mean (resp. Median) : mean (resp. median) of the ratio* CMLS *statistic /* DMLS *statistic.*

- Since CMLS has the flexibility to select pivots in the constraint matrix it may not be critical to know the metric of the entries that belong to a fairly dense row or column. That is why our CMLS implementation can easily avoid metric updates of such entries in the **C** matrix.
- Furthermore, supervariables have been generalized in our context resulting in separated row and column supervariables. This feature helps CMLS exploit the unsymmetric structure of the matrix in a more efficient way.

We then see in Table 5.3 that the decrease in fill-in and in the number of operations performed during the factorization phase leads to a decrease in the factorization time. The reductions in factorization time are slightly smaller than those in the number of operations (the average reduction in the number of operations is around 20%). This is because sparser factors often lead to smaller full blocks for which basic linear algebra kernels are slower. We observed that the flop rate of MA41_UNS tends to be smaller with CMLS than with DMLS: the average flop rate is nearly 1.02 GFlops with the DMLS ordering whereas it is around 0.99 GFlops with the CMLS ordering.

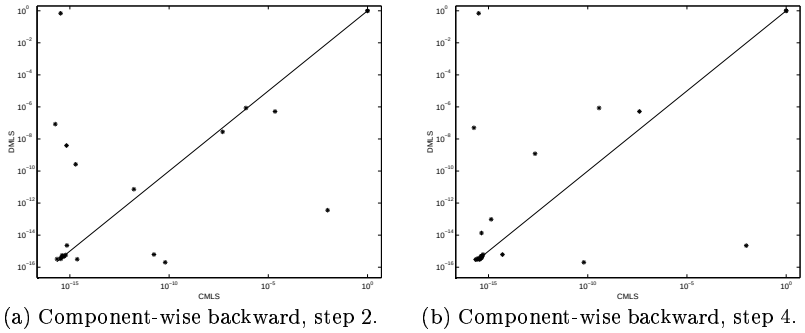
5.4. Impact of the hybrid strategies on SuperLU_DIST. We now study the numerical behaviour of SuperLU_DIST using CMLS ordering. Because of the static pivoting strategy used during factorization, SuperLU_DIST is expected to be numerically more sensitive than MA41_UNS to the use of hybrid strategies in pivot selection, and iterative refinement may be required to obtain an accurate solution, as was observed in [4]. We thus analyze the component-wise backward error of the solution [8] during iterative refinement. Note that one step of iterative refinement costs at least as much as one forward and backward substitution. The cost of the solution phase is closely related to the number of steps of iterative refinement. In the hybrid strategy (see Section 4.1), a relative threshold is set to avoid the selection of small pivots in **C**. This was chosen to be 0.01 in all our experiments.

	ordering time			factorization time			solution time		
Matrix	CMLS	DMLS	ratio	CMLS	DMLS	ratio	CMLS	DMLS	ratio
av41092	3.38	2.72	1.24	1.95	2.98	0.65	0.042	0.051	0.82
g7jac200sc	27.13	9.96	2.72	23.37	25.63	0.91	0.125	0.135	0.92
g7jac180sc	24.34	8.11	3.00	22.15	24.04	0.92	0.113	0.119	0.94
jan99jac120sc	5.29	3.01	1.75	1.48	2.04	0.72	0.037	0.043	0.86
jan99jac100sc	4.16	2.03	2.04	1.14	1.83	0.62	0.028	0.031	0.90
lhr34c	5.01	2.27	2.20	1.22	0.95	1.28	0.041	0.037	1.10
lhr71c	11.19	5.13	2.18	3.16	2.33	1.35	0.096	0.084	1.14
mark3jac120sc	8.26	3.59	2.30	5.88	4.90	1.20	0.069	0.070	0.98
mark3jac140sc	9.84	4.18	2.35	6.77	5.85	1.15	0.083	0.081	1.02
bayer01	1.66	1.19	1.39	0.36	0.49	0.73	0.037	0.039	0.94
sinc18	25.63	12.66	2.02	33.72	32.10	1.05	0.079	0.077	1.02
sinc15	10.89	4.60	2.36	10.98	10.79	1.01	0.040	0.040	1.00
Zhao2	2.22	0.93	2.38	5.67	6.97	0.81	0.052	0.053	0.98
mult_dcop_03	0.86	0.42	2.04	0.22	0.25	0.88	0.014	0.013	1.07
twotone	3.59	2.28	1.57	6.05	7.06	0.85	0.094	0.109	0.86
onetone1	1.52	0.50	3.04	1.25	1.70	0.73	0.026	0.030	0.86
torso1	14.49	70.20	0.20	15.87	25.25	0.62	0.180	0.191	0.94
bbmat	40.88	14.07	2.90	41.42	48.74	0.84	0.190	0.184	1.03
shermanACb	0.31	0.14	2.21	0.10	0.11	0.90	0.009	0.009	1.00
Mean/Median			2.1/2.2			0.91/0.88			0.97/0.98

TABLE 5.4

MA41_UNS *ordering, factorization and solution time (in seconds). ratio : ratio of* CMLS *statistic /* DMLS *statistic. Mean (resp. Median) : mean (resp. median) of the ratio* CMLS *statistic /* DMLS *statistic.*

Table 5.5 shows that SuperLU_DIST often does not compute an accurate solution if the CMLS ordering is obtained with a structural metric (compare the number of entries with $\star$ in columns STR and HYB). With the hybrid strategy to select pivots, some more pivots are postponed by CMLS because of their numerical values. We observe that this often results in an increase in the fill-in in the factor with respect to a structural metric (compare columns STR and HYB) but improves the numerical reliability of the CMLS pivot sequence. Note that we only report the median values because large variations of gains perturb the average statistics.



(a) Component-wise backward, step 2. (b) Component-wise backward, step 4.

FIG. 5.1. SuperLU_DIST *component-wise backward error during iterative refinement.*

Figure 5.1 compares the component-wise backward errors during iterative refinement with the CMLS and DMLS orderings (results after 2 and 4 steps). In each plot, a data point above the diagonal corresponds to a matrix for which CMLS performs better in terms of component-wise backward error. Is it clear that using the CMLS ordering improves the numerical behavior of SuperLU_DIST. There are still two

	Size of factors				Number of operations			
Matrix	STR	ratio	HYB	ratio	STR	ratio	HYB	ratio
av41092	*5773	0.72	*5993	0.74	1.12e+09	0.46	1.28e+09	0.52
g7jac200sc	*20394	0.92	20404	0.93	1.08e+10	1.18	1.11e+10	1.21
g7jac180sc	*19537	1.01	17426	0.90	1.26e+10	1.47	8.75e+09	1.02
jan99jac120sc	1923	0.82	1923	0.82	2.57e+08	0.64	2.57e+08	0.64
jan99jac100sc	1532	0.87	1532	0.87	2.04e+08	0.80	2.04e+08	0.80
lhr34c	*3342	1.19	3645	1.30	2.80e+08	2.09	4.19e+08	3.13
lhr71c	*7241	1.15	9434	1.50	7.25e+08	1.73	2.07e+09	4.93
mark3jac120sc	10847	1.04	10624	1.02	4.84e+09	1.17	4.29e+09	1.04
mark3jac140sc	*12552	1.01	12673	1.02	5.56e+09	1.13	5.53e+09	1.12
bayer01	909	0.66	909	0.66	1.80e+07	0.37	1.80e+07	0.37
sinc18	*24234	0.84	26702	0.93	3.46e+10	0.82	4.18e+10	1.00
sinc15	*11768	0.88	13447	1.00	1.17e+10	0.86	1.50e+10	1.10
Zhao2	*10954	0.87	*11174	0.89	5.94e+09	0.78	6.36e+09	0.84
mult_dcop_03	524	1.40	520	1.39	1.45e+07	4.84	1.30e+07	4.35
twotone	7118	0.90	7030	0.89	4.45e+09	1.06	4.38e+09	1.05
onetone1	2579	0.93	3020	1.09	7.98e+08	0.83	1.05e+09	1.10
torso1	*30156	0.88	*29455	0.86	2.35e+10	1.05	1.98e+10	0.88
fidapm11	*20777	0.99	21373	1.02	1.31e+10	0.96	1.41e+10	1.04
bbmat	36522	0.70	43074	0.82	2.07e+10	0.33	2.92e+10	0.47
shermanACb	342	0.85	347	0.87	1.62e+07	0.64	1.74e+07	0.69
cavity16	*321	0.77	330	0.79	1.86e+07	0.57	1.91e+07	0.58
INV-EXTRUSION-1	*25550	1.06	25973	1.08	2.15e+10	1.19	2.26e+10	1.25
MIXING-TANK	*44762	1.07	41760	1.00	7.54e+10	1.18	6.50e+10	1.02
Median		0.89		0.92		0.96		1.01

TABLE 5.5

SuperLU_DIST *size of the factors (in thousands) and number of operations (in millions).* STR*: the pivots are selected according to the* AMFI *structural metric.* HYB*: the pivots are selected using a hybrid strategy. ratio : ratio of* CMLS *statistic /* DMLS *statistic. Median : median value of the ratio* CMLS *statistic /* DMLS *statistic. *: after iterative refinement, the backward error is greater than* 10^{-8}.

matrices (av41092 and Zhao2) for which, with either CMLS or DMLS ordering, iterative refinement does not converge to an accurate solution (upper right corner). The torso1 matrix is the only one for which DMLS approach succeeds whereas CMLS approach fails (bottom right corner). There are four matrices in the upper left corner (lhr34c, lhr71c, mult_dcop3 and fidapm11) for which the backward error of SuperLU_DIST combined with DMLS remains larger than 10^{-8} whereas SuperLU_DIST combined with CMLS converges in less than four iterations. It is interesting to observe that the only matrices for which CMLS with the hybrid strategy leads to significantly more fill-in in the factors are the lhr34c, lhr71c and mult_dcop3 matrices on which DMLS did not converge after iterative refinement (and independently of the number of steps). Note finally that, with CMLS, on all problems except three we obtain an accurate solution (backward error smaller than 10^{-8}) with four steps of iterative refinement whereas, with DMLS, iterative refinement did not converge on six matrices.

6. Concluding remarks. The originality of the CMLS algorithm relies on its ability to compute an unsymmetric permutation with the following goals in mind: to reduce the fill-in in the factors and to preselect numerically good pivots for the factorization. It is based on a constraint matrix which contains the candidate pivots and a quotient graph that is used to compute the structural metrics. The CMLS algorithm can be used to design a family of orderings that can address a large class of problems. The main results and the properties of the algorithm are summarized as follows:

- Significant reductions in terms of fill-in (13%) and flops (20%) have been obtained with the structural strategy to select the pivots.
- Using structural metrics to select the pivots does not affect the numerical behaviour of the `MA41_UNS` solver.
- On numerically difficult problems, `CMLS` can be used to improve the accuracy of `SuperLU_DIST` and reduce the number of steps of iterative refinement during the solution phase.
- Our generalized supervariables could be used in the context of `DMLS` to also improve the metric computation.

One indirect but important consequence of our work is that we do not need to limit our pivot choice to a maximum weighted transversal of the original matrix. Preliminary experiments have shown that the maximum weighted matching can in fact be substituted by a simpler structural maximum transversal during the preprocessing phase (Step 1 as defined in Section 2). One possible direction for future work could then be to design a parallel version of the preprocessing phase.

Furthermore, the constraint matrix $\mathbf{C}$ contains the information of an incomplete factorization. We intend to use it as a preconditioner and to compare its quality and cost with existing incomplete $\mathbf{LU}$ factorizations.

Finally in our paper we have focused on local strategies and on very unsymmetric matrices. We also did experiments to compare our algorithms with global strategies such as nested dissection and observed that our approach was generally better on this set of test matrices. Combining our numerically based local heuristics with structurally based global strategies is another interesting direction for future work.

7. Acknowledgments. We want to thank Cleve Ashcraft and the two other anonymous referees for their useful and constructive indications on how to improve the presentation of this paper.

REFERENCES

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[2] P. R. Amestoy and I. S. Duff, *Vectorization of a multiprocessor multifrontal code*, International Journal of Supercomputer Applications, 3 (1989), pp. 41–59.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[4] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li, *Analysis and comparison of two general sparse solvers for distributed memory computers*, ACM Transactions on Mathematical Software, 27 (2001), pp. 388–421.

[5] P. R. Amestoy, X. S. Li, and E. Ng, *Diagonal Markowitz scheme with local symmetrization*, Tech. Rep. RT/APO/03/5, ENSEEIHT-IRIT, October 2003. Also appeared as Lawrence Berkeley Lab report LBNL-53854.

[6] P. R. Amestoy, X. S. Li, and S. Pralet, *Constrained Markowitz with local symmetrization*, Technical Report RT/APO/04/05, ENSEEIHT-IRIT, Toulouse, France, 2004. Also appeared as Lawrence Berkeley Lab report LBNL-56861 and CERFACS report TR/PA/04/137.

[7] P. R. Amestoy and C. Puglisi, *An unsymmetrized multifrontal LU factorization*, SIAM Journal on Matrix Analysis and Applications, 24 (2002), pp. 553–569.

[8] M. Arioli, J. Demmel, and I. S. Duff, *Solving sparse linear systems with sparse backward error*, SIAM Journal on Matrix Analysis and Applications, 10 (1989), pp. 165–190.

[9] C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM Journal on Matrix Analysis and Applications, 16 (1995), pp. 1404–1411.

[10] C. Ashcraft and R. G. Grimes, *SPOOLES: An object oriented sparse matrix library*, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, March 22–24, 1999.

[11] T. A. Davis, *University of Florida sparse matrix collection*, 2002. `http://www.cise.ufl.edu/research/sparse/matrices`.

[12] T. A. Davis, *Algorithm 832: UMFPACK V4.3 — an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, 30 (2004), pp. 196–199.

[13] 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.

[14] 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.

[15] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.

[16] 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.

[17] ———, *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.

[18] I. S. Duff and J. K. Reid, *A comparison of sparsity orderings for obtaining a pivotal sequence in Gaussian elimination*, Journal of the Institute of Mathematics and its Applications, 14 (1974), pp. 281–291.

[19] ———, *MA27—a set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Technical Report R.10533, AERE, Harwell, England, 1982.

[20] ———, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

[21] ———, *The multifrontal solution of unsymmetric sets of linear systems*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 633–641.

[22] ———, *MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems*, Tech. Rep. RAL 95-001, Rutherford Appleton Laboratory, 1995.

[23] A. George and J. W. H. Liu, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Transactions on Mathematical Software, 6 (1980), pp. 337–358.

[24] ———, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ., 1981.

[25] 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. 90–111.

[26] A. Gupta, *Recent advances in direct methods for solving unsymmetric sparse systems of linear equations*, ACM Transactions on Mathematical Software, 28 (2002), pp. 301–324.

[27] X. S. Li and J. W. Demmel, *A scalable sparse direct solver using static pivoting*, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, Texas, March 22–24 1999.

[28] ———, *SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Transactions on Mathematical Software, 29 (2003).

[29] J. W. H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.

[30] E. Ng and P. Raghavan, *Performance of greedy heuristics for sparse Cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 902–914.

[31] G. Pagallo and C. Maulino, *A bipartite quotient graph model for unsymmetric matrices*, in Lecture Notes in Mathematics 1005, Numerical Method, Springer-Verlag, New York, 1983, pp. 227–239.

[32] S. Pralet, *Constrained orderings and scheduling for parallel sparse linear algebra*, PhD thesis, Institut National Polytechnique de Toulouse, Sept 2004. Available as CERFACS technical report, TH/PA/04/105.

[33] E. Rothberg and S. C. Eisenstat, *Node selection strategies for bottom-up sparse matrix ordering*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 682–695.

[34] A. F. Van Der Stappen, R. H. Bisseling, and J. G. G. Van De Vorst, *Parallel sparse LU decomposition on a mesh network of transputers*, SIAM Journal on Matrix Analysis and Applications, 14 (1993), pp. 853–879.

[35] R. C. Whaley and A. Petitet, *Minimizing development and maintenance costs in supporting persistently optimized BLAS*, Software: Practice and Experience, 35 (2005), pp. 101–121. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

[36] R. C. Whaley, A. Petitet, and J. J. Dongarra, *Automated empirical optimization*

of software and the ATLAS project, Parallel Computing, 27 (2001), pp. 3–35. Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).

[37] Z. Zlatev, *On some pivotal strategies in Gaussian elimination by sparse technique*, SIAM Journal on Numerical Analysis, 17 (1980), pp. 18–30.