UNSYMMETRIC ORDERING USING A CONSTRAINED MARKOWITZ SCHEME [*]

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

Abstract. We consider the *LU* factorization of unsymmetric sparse matrices using a three-phase approach (analysis, factorization and triangular solution). Usually the analysis phase first determines a set of potentially good pivot and then orders this set of pivots to decrease the fill-in in the factors. In this paper, we present a preprocessing algorithm that simultaneously achieves the objectives of selecting numerically good pivots and preserving the sparsity. We describe the algorithmic properties and difficulties in 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 obtain the average gains of 14% in the factorization time, of 12% in the size of the *LU* factors, and of 21% in the number of operations performed in the factorization phase.

1. Introduction. We consider the *LU* factorization of a unsymmetric sparse matrix $\mathbf{A}$ based on a *three-phase approach*. It includes an *analysis* phase, a *numerical factorization* phase, and a *triangular solution* phase. In this class of methods, without loss of generality, we will consider the multifrontal and the supernodal algorithms (see for example [2, 8, 12, 17, 19]). The analysis phase transforms $\mathbf{A}$ into $\bar{\mathbf{A}}$ with better properties for sparse factorization. It exploits the structural information to reduce the number of fill-ins in the *LU* factors and exploits the numerical information to reduce the number of numerical pivotings needed during factorization. Two separate treatments are commonly used in sequence for these two objectives. Firstly, scaling and maximum transversal algorithms are used to transform $\mathbf{A}$ into $\mathbf{A}_1$ with large entries in magnitude on the diagonal. Secondly, a symmetric fill-reducing ordering, which preserves the large diagonal, is used to permute $\mathbf{A}_1$ into $\mathbf{A}_2$ so that the factors of $\mathbf{A}_2$ are sparser than those of $\mathbf{A}_1$. Note that during factorization, numerical instabilities can still occur and will be handled either by partial pivoting resulting in extra fill-ins in the factored matrices or by static pivoting resulting in a potentially less accurate factorization. This approach has two drawbacks:

- The numerical treatment forces the fill-reducing ordering to restrict pivot selection to the diagonal of A_1, and so to compute a symmetric permutation,
- The ordering phase does not have numerical information to select pivots.

To reduce the fill-in in the factors and improve the numerical quality of the preselected pivots, we describe in this paper, a family of orderings that can select off-diagonal pivots based on 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

[*] 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.

[†] 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@cerfacs.fr, CERFACS, 42, av. G. Coriolis, 31057 Toulouse Cedex 01, France.

information in **C**. The **C** matrix serves as a **constraint matrix** for the pivot selection. The new algorithm is referred to as *constrained Markowitz with local symmetrization* (or CMLS). In this context, the use of local symmetrization introduced in [4] will be explained and justified in Section 3.2.

We will focus on two state-of-the-art direct solvers representative of this class: the multifrontal code MA41_UNS [2, 5] and the supernodal code SuperLU_DIST [27, 28]. For the MA41_UNS multifrontal code, 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. Postponing the elimination of a pivot will then result in an increase in the size of the **LU** factors estimated during analysis and in an increase in the number of operations. In the case of the supernodal code, SuperLU_DIST, the distributed memory version uses a right-looking formulation which, having computed the factorization of a block of columns, then immediately sends the data to update the block columns in the trailing submatrix. A static pivoting strategy is used and we keep the pivotal sequence chosen in the analysis. The result is that iterative refinement may be needed to improve the solution.

In practice, it has been observed in [3] that setting large entries on the diagonal, based on [14], one can significantly reduce the number of numerical problems for both solvers. Nevertheless, the static approach can still fail on some problems and relies more on iterative refinement in the solution phase. In this context, the problem of finding a good ordering for preserving the diagonal entries (symmetric ordering) and taking into account the asymmetry of the matrix was studied in [4]. In this paper the work described in [4] (referred to as DMLS) has been extended and generalized as follows. Firstly, we do not limit our choice of pivots to a transversal of the original matrix. Secondly, at each step of elimination, the pivot is chosen within a constraint matrix **C**, which is updated at each step. Thirdly, we may also consider numerical values and numerical updates during the pivot selection. To do so, we compute an incomplete factorization of the constraint matrix. Finally, because of all the new concepts introduced in this approach, we had to introduce new algorithms, revisit the existing ones and introduce new data structures to implement our algorithms.

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 **A**, our unsymmetric ordering consists of two main steps:

- **Step 1:** Based on a numerical pre-treatment of the matrix **A**, we extract a set of numerically acceptable pivots, referred to as the **constraint matrix C**. We have $\mathcal{P}attern(\mathbf{C}) \subset \mathcal{P}attern(\mathbf{A})$, 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 more precisely these 2 steps, 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 **M** can be permuted to have nonzeros on the diagonal then **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 $\mathcal{M} \subset E$ such that for all vertices $v \in V$, 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 say also 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 over-complicate our notations and our comments.

2.1. Step 1: Numerical preprocessing. The objective of this preprocessing step is to extract the most significant (structurally and numerically) entries of 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 $\mathbf{A} \leftarrow \mathbf{D}_r\mathbf{A}\mathbf{D}_c$. Secondly, a *constraint matrix* $\mathbf{C}$ can be constructed from $\mathbf{A}$ such that $\mathcal{P}attern(\mathbf{C}) \subset \mathcal{P}attern(\mathbf{A})$ 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 $\mathcal{M}$. Note that $\mathcal{M}$ is also a perfect transversal of $\mathbf{A}$.

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 $\mathcal{P}attern(\mathbf{C}^1) \subset \mathcal{P}attern(\mathbf{A}^1)$. 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 row and 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 the pattern of $\bar{\mathbf{C}}^k$, where $\bar{\mathbf{C}}^k$ is defined as the reduced matrix after the elimination of pivot p^k in $\mathbf{C}^k$. Note that $\bar{\mathbf{C}}^k$ includes the fill-in resulting from the elimination of p^k in $\mathbf{A}^k$. This implies that $\mathcal{P}attern(\mathbf{C}^{k+1}) \subset \mathcal{P}attern(\mathbf{A}^{k+1})$. 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 $\mathbf{C}$:

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

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

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

$$\mathbf{C} \text{ must remain structurally nonsingular,} \tag{2.1}$$

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

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. The structure of an unsymmetric matrix can be represented as a bipartite graph, and Gaussian elimination can be efficiently modeled by a bipartite quotient graph [31]. 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 [4], a technique that simplifies the bipartite quotient graph implementation. Note that we use calligraphic letter for notations related to quotient graphs.

3.1. Bipartite graph. Let $\mathbf{C} = (c_{ij})$ be a matrix and $G_C = (V_r, V_c, E)$ be its associated bipartite graph. Let R_i denotes 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 (i,j) is eliminated, a new matrix, referred to as the reduced matrix $\bar{\mathbf{C}}$ is computed. $\bar{\mathbf{C}}$ is obtained from $\mathbf{C}$ by removing row i and column j and by adding the Schur complement entries. In terms of graph manipulations, this elimination adds edges in the bipartite graph of $\mathbf{C}$ to connect all the rows adjacent to j to all the columns adjacent to i. This set of connected rows and columns is referred to as a **bi-clique**.

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

More formally, the changes of the bipartite graph from G_{C^k} to $G_{C^{k+1}}$ are

$$R_i^{k+1} = (R_i^k \cup R_{row_p}^k) \setminus \{col_p\} \text{ for } i \in C_{col_p}^k \ ,$$

$$C_j^{k+1} = (C_j^k \cup C_{col_p}^k) \setminus \{row_p\} \text{ for } j \in R_{row_p}^k$$

and remove row_p and col_p from the vertex sets of $G_{C^{p+1}}$. When it is clear from the context, we will use the notation L_p in place of $R_{row_p}^p$ and U_p in place of $C_{col_p}^p$.

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, the entries to the Schur complement matrix which may be costly to update and to store. It has been shown that the quotient graphs can be used to efficiently model the factorization of symmetric matrices [18, 22]. 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 $\mathbf{A}$. 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 the notations that will be used to describe our algorithms.

Let $\mathcal{G}^k$ 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 $\mathcal{G}^1$ is identical 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 $\mathcal{G}^k$. 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 $\mathcal{G}^k = (\mathcal{V}_r \cup \overline{\mathcal{V}}_r, \mathcal{V}_c \cup \overline{\mathcal{V}}_c, \mathcal{E} \cup \overline{\mathcal{E}})$. 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 $\mathcal{E} \subset (\mathcal{V}_r \times \mathcal{V}_c)$ whereas $\overline{\mathcal{E}}$ is such that $\overline{\mathcal{E}} \subset (\mathcal{V}_r \times \overline{\mathcal{V}}_c) \cup (\overline{\mathcal{V}}_r \times \mathcal{V}_c) \cup (\overline{\mathcal{V}}_r \times \overline{\mathcal{V}}_c)$. With our definitions (r, c) is a nonzero entry in the reduced matrix at step k if and only if there exists a path joining r and c 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 $e_i = (r_{e_i}, c_{e_i}), 1 \leq i \leq l$ 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 $\mathcal{G}^k$ [31] and in particular through more than one coupled element. For example in Figure 3.1, we assume that the entry (row_p, c_{e2}) corresponds to the fill-in due to the elimination of element e_1 and is implicitly represented by an edge between variable row_p and element c_{e1} in the quotient graph. In this case we know that the row structure of row_p contains the row structure of $e2$ because of the path $row_p \rightarrow c_{e1} \rightarrow r_{e1} \rightarrow c_{e2} \rightarrow r_{e2}$.

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 $p = (row_p, col_p)$ is selected, if there exists a cycle of the form $row_p \rightarrow c_{e_1} \rightarrow r_{e_1} \ldots \rightarrow c_{e_l} \rightarrow r_{e_l} \rightarrow col_p \rightarrow row_p$ (referred to as a **strongly connected component** in [31]) then, except for row_p and col_p, the row and column elements in the cycle are not needed any more to retrieve the structure of the remaining variables. Firstly, all the row (resp. column) variables adjacent to these column (resp. row) elements will be included in the adjacency of col_p (resp. row_p), and secondly, the row (resp. column) variables which were adjacent to one of the removed elements will be adjacent to col_p (resp. row_p). This process will be called **element absorption** and is illustrated in Figure 3.1.

To avoid long search paths, we decided to relax the element absorption rule. 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 [4]. It implies that the resulting quotient graph $\mathcal{G}^k$ at step k models only an approximation of the structure of the reduced submatrix. It has been shown in [4] 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 $\mathcal{G}^1$. 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.

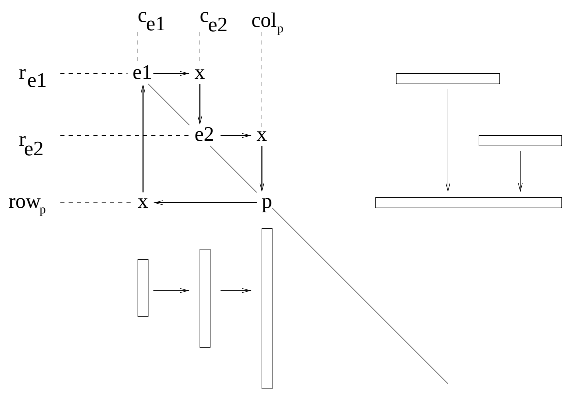


FIG. 3.1. *Illustration of a cycle ($row_p \to c_{e_1} \to r_{e_1} \to c_{e_2} \to r_{e_2} \to col_p \to row_p$).*

To simplify the description of how the bipartite quotient graph is modified at each elimination step, we define $\overline{\mathcal{V}} \subset (\overline{\mathcal{V}_r} \times \overline{\mathcal{V}_c})$ 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 row (resp. column) variables adjacent in $\mathcal{G}^k$ to the row (resp. column) element of a pivot $p = (row_p, col_p)$. Thanks to local symmetrization, the concept of absorption can be extended to coupled elements: an element $e = (r_e, c_e)$ such that $(row_p, c_e) \in \overline{\mathcal{E}}$ **or** $(r_e, col_p) \in \overline{\mathcal{E}}$ 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 [5].

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 $\mathcal{G}$ 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. $\mathcal{C}_j$). At the beginning $\mathcal{R}_i = \mathcal{C}_j = \emptyset$ and $\mathcal{A}_{i*} = \mathcal{A}_{i*}^{(0)}$ and $\mathcal{A}_{*j} = \mathcal{A}_{*j}^{(0)}$ 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 (computation of $\mathcal{L}_p$ and $\mathcal{U}_p$). The variable lists $\mathcal{A}_{i*}$ and $\mathcal{A}_{*j}$ can also be pruned. Indeed, the edges in $\mathcal{G}$ 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 $(i, j) \in \mathcal{V}_r \times \mathcal{V}_c$ is selected as the next pivot we build the element $p =$

$(i,j) \in \mathcal{G}$ such that:

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

and

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

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 col_{p1} is adjacent to $e1$, local symmetrization adds the virtual $Sp1$ entry so that the row structure of $p1$ contains $\mathcal{U}_{e1}$. Let $\mathcal{F}_p = \mathcal{C}_j \cup \mathcal{R}_i$ be the set of elements adjacent to the current pivot. Note that when it is clear from the context, $\mathcal{F}$ will also refer to the elements adjacent to the current variable. The elements in $\mathcal{F}_p$ are absorbed by p and can be removed from the quotient graph. For each column variable j_1 in $\mathcal{U}_p$ (resp. i_1 in $\mathcal{L}_p$) the element p is added to its adjacency and the elements in $\mathcal{F}_p$ are removed: $\mathcal{C}_{j_1} \leftarrow (\mathcal{C}_{j_1} \setminus \mathcal{F}_p) \cup \{p\}$ (resp. $\mathcal{R}_{i_1} \leftarrow (\mathcal{R}_{i_1} \setminus \mathcal{F}_p) \cup \{p\}$). The structure of column j_1 of the factors in the reduced matrix is then given by $\mathcal{A}_{*j_1} \cup \bigcup_{e \in \mathcal{C}_{j_1}} \mathcal{L}_e$. The structure of row i_1 of the factors is $\mathcal{A}_{i_1*} \cup \bigcup_{e \in \mathcal{R}_{i_1}} \mathcal{U}_e$.

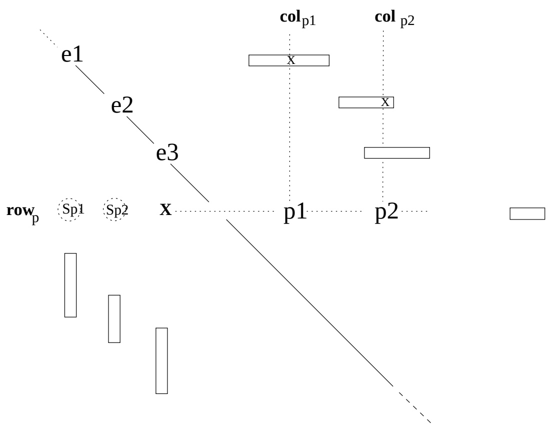


FIG. 3.2. *Influence of local symmetrization on the pivot structure*

Note that, although the above structural changes of the reduced submatrix are correct, they should not be used to compute the structural metrics. Indeed, if (i_1, j_1) is 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 row_p, $p1 = (row_p, col_{p1})$ and $p2 = (row_p, col_{p2})$. We assume that all the elements in $\mathcal{G}$ adjacent to $p1$ and $p2$ are indicated in the figure. The structure of row_p is then given by $\mathcal{A}_{row_p*} \cup \mathcal{U}_{e_3}$. This however does not give enough information on the structure of row_p if either $p1$ or $p2$ were selected as the next pivot. If $p1$ were the next pivot then the structure of row_p would be given by $\mathcal{U}_{p1} = \mathcal{A}_{i*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_1}$ because of the locally symmetrized entry S_{p1}. If $p2$ were the next pivot then the structure of row_p

Bipartite graph G		Bipartite quotient graph $\mathcal{G}$	
$G_C = (V_r, V_c, E)$		$\mathcal{G}^k = (\mathcal{V}_r \cup \overline{\mathcal{V}}_r, \mathcal{V}_c \cup \overline{\mathcal{V}}_c, \mathcal{E} \cup \overline{\mathcal{E}})$	
R_i	structure of row i	$\mathcal{R}_i$	elements adjacent to row i
		$\mathcal{A}_{i*}$	variables adjacent to row i
A_{i*}	initial structure of row i	$\mathcal{A}_{i*}^{(0)}$	initial structure of row i
C_j	structure of column j	$\mathcal{C}_j$	elements adjacent to column j
		$\mathcal{A}_{*j}$	variables adjacent to column j
A_{*j}	initial structure of column j	$\mathcal{A}_{*j}^{(0)}$	initial structure of column j
U_p	row structure of the p^{th} pivot	$\mathcal{U}_p$	row structure of the p^{th} pivot
L_p	column structure of the p^{th} pivot	$\mathcal{L}_p$	column structure of the p^{th} pivot
		$\mathcal{F}$	elements that are adjacent to row i or column j for a pivot (i,j) ($\mathcal{F} = \mathcal{R}_i \cup \mathcal{C}_j$)

TABLE 4.1
Notations used for bipartite graph and bipartite quotient graph.

would be given by $\mathcal{U}_{p2} = \mathcal{A}_{i*} \cup \mathcal{U}_{e_3} \cup \mathcal{U}_{e_2}$ 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 $\mathcal{G}$ before the pivot selection, we should anticipate its effect on the metrics used to select the best pivot between p_1 and p_2.

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 and the metric 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 (for example, a numerical value and a structural metric) that will be used to select a pivot. On the other hand, in order to have a fast computation of the structural metrics 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 two graphs 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 $\mathcal{G}$ 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 $\mathbf{C}^k$, the structural metric relative to $\mathcal{G}^k$. Section 4.5 finally explains how supervariables are defined and used in our context.

4.1. Pivot selection.

At each step, the pivot with minimal metric is selected. The choice of a metric implies the underlying algorithmic strategy. We say that we use **structural strategies** in our algorithms when the entries are selected with respect to only the structural metrics whereas the **hybrid strategies** correspond to the combination of structural and numerical metrics.

Using doubly-linked lists and a hash function based on the value of the structural metric provides a direct access to the entry in $\mathbf{C}$ that has the minimum structural metric in G_C.

Furthermore, as it is often the case in sparse matrix factorization, we may want to preserve the sparsity of the factors while controlling the growth in the factors. Numerical thresholds are introduced to give freedom for the pivot selection to balance numerical precision with sparsity preservation. For each entry $(i,j) \in \mathbf{C}^k$ we define its numerical metric: $v(i,j) = |c_{ij}|/||c_{.j}||_\infty$. A pivot in position (i,j) 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 [11] the authors proposed to visit the entries of a fixed number of columns using the Zlatev-style search [34]. We use a slightly different algorithm to limit our search. At each step of the ordering, we look for the best entry $p = (row_p, col_p)$ within a subset (say S) of the entries in the bipartite graph G_C. The subset S is defined by two threshold parameters $MS > 0$ and $NCOL \geq 0$ 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 different 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 $= S \setminus MS$-set. This implies that if $NCOL > 0$, the $NCOL$-set will guarantee that the numerically best pivots in the first $NCOL$ columns will belong to S. As for the structural strategies, it is straightforward to access entries in the MS-set and their corresponding columns using the doubly-linked lists and a hash function.

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 $NCOL > 0$ then we are sure that at least $NCOL$ entries will be numerically acceptable since $\tau \leq 1$. In this case, however, all the entries of S need to be considered to obtain the minimum on S according to our hybrid strategy. Finally if $NCOL = 0$ 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.

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 the Properties (2.1) and (2.2) hold.

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

- **MATCHUPDATE** will refer to the strategy that performs incomplete Gaussian elimination to only preserve the perfect matching property (2.1). Let $p = (row_p, col_p)$ be the current pivot. Let $(row_p, match\_col)$ and $(match\_row, col_p)$ be the matched entries of $\mathbf{C}$ in row row_p and column col_p, respectively. That is, $(row_p, match\_col) \in \mathcal{M}$ and $(match\_row, col_p) \in \mathcal{M}$. If these entries are the same (*i.e.*, (row_p, col_p) is a matched entry), nothing needs to be done to maintain Property (2.1). Otherwise entry $(match\_row, match\_col)$ is added 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.
- **TOTALUPDATE** will refer to the strategy which performs all the updates in $\mathbf{C}$ (i.e., $\mathbf{C}^{k+1} = \bar{\mathbf{C}}^k$).

In practice, a mixed strategy, exploiting both MATCHUPDATE and TOTALUPDATE will be used for the experiments. At each step, TOTALUPDATE strategy is used except when we want to limit the memory or the time complexity, then MATCHUPDATE strategy is used. See [32] for further details.

4.3. Update of the bipartite quotient graph $\mathcal{G}$. 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 $\mathcal{G}^k$

```
  Let $p = (row_p, col_p)$ be the current pivot at step $k$ and $\mathcal{F}_p = \mathcal{R}_{row_p} \cup \mathcal{C}_{col_p}$.
  if $\mathcal{U}_p \neq \emptyset$ and $\mathcal{L}_p \neq \emptyset$ then
    for each row $i \in \mathcal{L}_p$ do
1     $\mathcal{A}_{i*} = (\mathcal{A}_{i*} \setminus \mathcal{U}_p) \setminus \{col_p\}$ /* 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     $\mathcal{A}_{*j} = (\mathcal{A}_{*j} \setminus \mathcal{L}_p) \setminus \{row_p\}$ /* 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
      $\mathcal{R}_i = (\mathcal{R}_i \setminus \mathcal{F}_p)$
      $\mathcal{A}_{i*} = \mathcal{A}_{i*} \setminus \{col_p\}$
    end for
    for each column $j \in \mathcal{U}_p$ do
      $\mathcal{C}_j = (\mathcal{C}_j \setminus \mathcal{F}_p)$
      $\mathcal{A}_{*j} = \mathcal{A}_{*j} \setminus \{row_p\}$
    end for
  end if
```

The "if" block of code 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 reducibility detection as done in the "else" block of Algorithm 4.1 impossible. Note that this additional pruning would have improved the accuracy of our structural metrics as explained in Section 4.4.

The "else" block of Algorithm 4.1 shows that the algorithm can exploit the fact that when only one of the $\mathcal{L}_p$ or $\mathcal{U}_p$ is empty we can prune the current pivot from the quotient graph $\mathcal{G}$ and therefore benefit from the reducibility of the input matrix. This feature of the CMLS algorithm will be justified in Section 4.3.2. We will also explain why it is correlated with the strategy used to prune variables.

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. *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 $\mathcal{P}attern(\mathbf{C}^{k+1}) \subset \mathcal{P}attern(\bar{\mathbf{C}}^k)$. 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, it is for sure that (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 [4]. This is illustrated in Figure 4.1. 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.

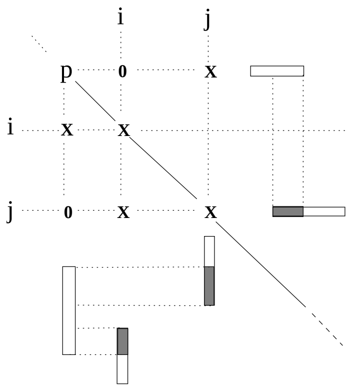


FIG. 4.1. *Illustration of two-way variable elimination.*

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. Consider Figure 4.1 again, if all three entries (i, i), (j, j) and (j, i) belong in $\mathbf{C}$, then we cannot prune all the shaded areas. This is because both (i, i) and (j, i) can be potential pivots from column i. If (i, i) is chosen as the pivot from that column, then the shaded area in column i can be pruned, because $i \in \mathcal{L}_p$ and local symmetrization is invoked. However, if (j, i) is selected as the pivot from that column, then since $j \notin \mathcal{L}_p$ and $i \notin \mathcal{U}_p$, the element p will not be used to build the row and column adjacency of (j, i), and the shaded area in column i cannot be pruned. Otherwise, we cannot retrieve those variables from anywhere. The same observation applies to the shaded area in row j if (j, i) instead of (j, j) is chosen as the pivot. 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 $\mathcal{U}_p = \emptyset$ (we may call it *strongly reducible*) or (2) $\mathcal{L}_p = \emptyset$ or $\mathcal{U}_p = \emptyset$ (we may call it *weakly reducible*). Ideally, we would like to remove p from the quotient graph $\mathcal{G}$ in both reducible cases. 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 $\mathcal{G}$. Therefore, pivot p can be removed from $\mathcal{G}$.

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 $\mathcal{A}^{(0)}_{*j}$ have been pruned under two-way variable elimination. Therefore, p must be used to retrieve those entries and cannot be removed from $\mathcal{G}$. This is illustrated in Figure 4.2 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.

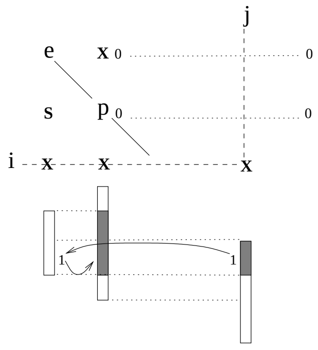


FIG. 4.2. *Effect of variable elimination in both directions on reducibility detection (S indicates the position of local symmetrization).*

In the rest of this section, we discuss the detection and/or the exploitation of the BTF (Block Triangular Form) of the initial matrix. We show that, under certain assumptions, using one-way variable elimination, we can exploit and detect part of the BTF.

DEFINITION 4.1. *We say that an ordering is* **BTF compatible** *when*

(1) it selects only pivots within the diagonal blocks of the BTF;

(2) it eliminates all the pivots in a block before processing another block;

(3) *the order in which the diagonal blocks are considered must be such that each selected block is not adjacent to any remaining block in either the row or the column direction.*

Note that condition (3) allows a block to be considered in the order of the BTF but also allows some slight variation of this form as will be shown in Figure 4.3.

PROPERTY 4.4. *If* `CMLS` *selects pivots with a BTF compatible ordering then the sparsity of the block triangular form will be fully preserved and* `CMLS` *will generate a forest in which each tree corresponds to an irreducible component of the BTF.*

This property is a consequence of Property 4.2. We illustrate in the following why condition (3) is needed in Definition 4.1. We also comment on the benefit of using one-way variable elimination when the matrices are reducible and on the adverse effect of two-way variable elimination for reducible matrices.

Let us assume that the constraint matrix $\mathbf{C}$ is restricted to the diagonal of $\mathbf{A}$ and that pivots are chosen down the diagonal in order. In this case, the two-way variable elimination could also be applied at each step of the ordering (see Property 4.1). Figure 4.3 shows a matrix with five full diagonal blocks of size k. The original matrix in BTF is shown on the left-hand side. The structure resulting from the use of a modified `CMLS` algorithm that would include two-way variables elimination is shown in the middle of the figure. We see that, with two-way variable elimination, local symmetrization interconnects all the diagonal blocks of the BTF and the BTF forest (independence between the diagonal blocks) is lost. In addition, because of the nonzero “s” between block (1) and block (2), the first row of block (2) is completely filled, which in turn will create fill-ins in all the lines of block (2) and in all the lines of the subsequent blocks.

With only one-way variable elimination, the last pivot of block (1) will be removed (see Property 4.2). The same holds for all subsequent blocks so that Algorithm 4.1 does not introduce any fill-in and produces a forest in which each tree corresponds to a diagonal block of the BTF matrix.

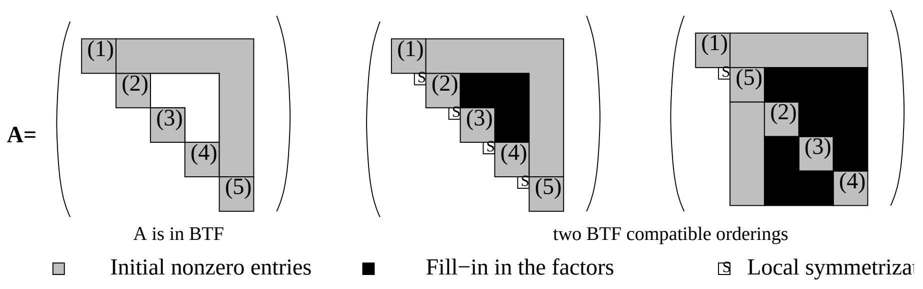


FIG. 4.3. *Illustration of two BTF compatible orderings of a matrix A in BTF. Using two-way variable elimination introduces fill-ins in the factors.*

We now assume that the blocks of the matrix have been symmetrically permuted in the order 1,5,2,3,4, see the right-hand side matrix in Figure 4.3. This ordering does not respect the BTF, however, it is BTF compatible thanks to the condition (3) of Definition 4.1. Property 4.2 can be applied to the last pivot of each block, since either $\mathcal{U}_p$ or $\mathcal{L}_p$ will be empty. The last pivot selected in each block will become the root of the tree associated with that block of the BTF matrix. Algorithm 4.1 thus fully exploits the BTF and preserves the sparsity of matrix. On the other hand, if we use two-way variable elimination, because of the “s” entry at position $(k+1,k)$, row

$k+1$ becomes completely filled and after eliminating the $(k+1)$st diagonal entry the remaining submatrix becomes also full.

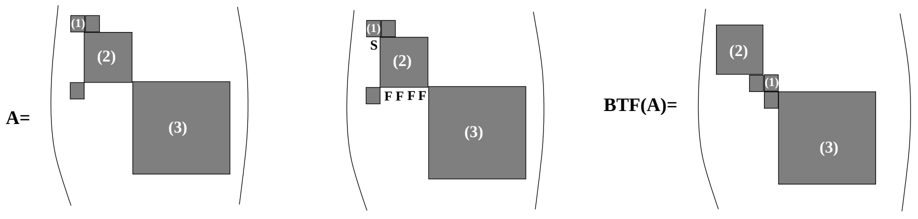


FIG. 4.4. *The initial matrix is not in BTF (left plot) and the ordering is not BTF compatible (factors in the middle plot). The ordering on the right is BTF compatible and preserves sparsity. F: fill-in. S: local symmetrization.*

Finally, we show in Figure 4.4 that when the ordering of the blocks does not satisfy condition (3) of Definition 4.1, CMLS algorithm may partially lose BTF. In this example, local symmetrization between blocks (1) and (2) will create a dependency between them and fills one rectangular block because of the propagation of this dependency. Block (3) will however remain an independent tree because the last pivot p in block (2) will have $\mathcal{U}_p = \emptyset$ and p will become the root of the tree containing blocks (1) and (2). Note that if two-way variable elimination were applied, the ordering would not have detected the independence of block (3).

4.4. Update of the structural metric. In this section, we describe two classes of local heuristics to estimate the structural quality of a pivot. In the preamble section, we first describe the common framework that is independent of the metric used. In Section 4.4.2, the approximate Markowitz cost [4] is briefly introduced. In Section 4.4.3, 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. Finally, in Section 4.4.4, we discuss the complexity of the algorithms used to update the structural metrics.

4.4.1. Preamble. Let us assume that the p^{th} pivot (row_p, col_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 since the structure of rows in $\mathcal{L}_p$ and columns in $\mathcal{U}_p$ might have changed. The size of this area is thus larger than the area involved in the update of the structure of $\mathbf{C}$ since columns (resp. rows) of $\mathcal{U}_p \setminus R_{row_p}$ (resp. of $\mathcal{L}_p \setminus C_{col_p}$) need also be considered. The algorithm to update the structural metrics will be 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 has been applied. In the following, the degrees, approximate degrees, fill-ins and approximate fill-ins are all related to this quotient graph structure. Note that, since for symmetric matrices our quotient graph becomes the standard symmetric quotient graph, all the discussions naturally apply to symmetric matrices.

Equations (3.1) and (3.2) that include local symmetrization could be used to compute the exact external degrees, but it would be costly. Instead, similar to AMD [1] and DMLS [4] algorithms, approximate row and columns external degrees can be computed. The AMD like approximate external row and columns degree, $amd_r(i,j)$ and $amd_c(i,j)$ respectively, are then defined by the following two equations:

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

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

As observed in [4], 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 as 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 shown in Section 4.3, the benefit of one-way variable elimination is to better exploit the BTF of the matrix.

4.4.2. Approximation of Markowitz cost. The approximation of the Markowitz cost comes directly from our approximation of the external row and column degrees. After eliminating the k^{th} pivot, we define

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

and

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

The metric associated with the approximate Markowitz cost is then defined as

$$metric^{(k+1)}(i,j) = \min \begin{cases} \overline{amd_r}(i,j) \times \overline{amd_c}(i,j) \\ metric^{(k)}(i,j) & +|\mathcal{U}_p \setminus j| \times \overline{amd_c}(i,j) \\ & +|\mathcal{L}_p \setminus i| \times \overline{amd_r}(i,j) \\ & -|\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i|. \end{cases} \tag{4.5}$$

Here, we use the convention that if (i,j) is a new entry in $\mathbf{C}$, then $metric^{(k)}(i,j)$ in equation (4.5) is set to $+\infty$. Note that during the update of the degrees, contrary to AMD, we do not use the values of approximate row and column degrees computed at the previous step. This could have been done but would have required us to store two other arrays (one for row degrees, one for column degrees) of size $|\mathbf{C}|$ since the approximate degree varies within a given row or column.

4.4.3. Approximation of the fill-in. With a minimum fill-in based metric, we want to estimate the new fill-in that would occur in the reduced matrix if an entry were selected as the next pivot. For the sake of completeness, one should mention that the fill-in metric of variables at distance two from the pivot might also vary. In this work, we will only consider variables adjacent to the pivot, *i.e.* at distance one, since results on symmetric matrices have shown that considering distance two variables significantly increases the complexity of the algorithm for relatively little gain in the quality of the ordering [30, 33].

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. We explain how to compute this new upper bound using already computed information and local correction terms.

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 $\mathcal{F}$. To simplify the notation we define $\tilde{\mathcal{L}_e} = (\mathcal{L}_e \setminus \mathcal{L}_p) \setminus \{i\}$, $\tilde{\mathcal{U}_e} = (\mathcal{U}_e \setminus \mathcal{U}_p) \setminus \{j\}$, $\tilde{\mathcal{A}_{*j}} = (\mathcal{A}_{*j} \setminus \mathcal{L}_p) \setminus \{i\}$, $\tilde{\mathcal{A}_{i*}} = (\mathcal{A}_{i*} \setminus \mathcal{U}_p) \setminus \{j\}$, $\hat{\mathcal{L}_e} = \mathcal{L}_e \setminus \{i\}$ and $\hat{\mathcal{U}_e} = \mathcal{U}_e \setminus \{j\}$. Thus a $\tilde{}$ means that we subtract the current pivot structure and the current variable and a $\hat{}$ means that we only subtract the current variable.

Let $d_r(i,j)$ and $d_c(i,j)$ denote the external row and column external degrees of entry (i,j) respectively. With our notation we have:

$$d_r(i,j) = |(\mathcal{U}_p \cup \mathcal{A}_{i*} \cup \bigcup_{e \in \mathcal{F}} \mathcal{U}_e) \setminus \{j\}| \leq |\hat{\mathcal{U}_p}| + |\tilde{\mathcal{A}_{i*}}| + |\bigcup_{e \in \mathcal{F}} \tilde{\mathcal{U}_e}| \tag{4.6}$$

$$d_c(i,j) = |(\mathcal{L}_p \cup \mathcal{A}_{*j} \cup \bigcup_{e \in \mathcal{F}} \mathcal{L}_e) \setminus \{i\}| \leq |\hat{\mathcal{L}_p}| + |\tilde{\mathcal{A}_{*j}}| + |\bigcup_{e \in \mathcal{F}} \tilde{\mathcal{L}_e}|. \tag{4.7}$$

Let $S(i,j)$ denote the union of the areas associated with all the elements adjacent to entry (i,j):

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

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

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

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

$$amd_r(i,j) amd_c(i,j) - |\hat{\mathcal{U}_p}||\hat{\mathcal{L}_p}| - S(i,j). \tag{4.8}$$

Let AS be an overestimation of area S,

$$AS(i,j) = \sum_{e \in \mathcal{F}} |(\hat{\mathcal{L}_e} \times \hat{\mathcal{U}_e}) \setminus (\mathcal{L}_p \times \mathcal{U}_p)| = \sum_{e \in \mathcal{F}} |\tilde{\mathcal{U}_e}||\hat{\mathcal{L}_e}| + |\tilde{\mathcal{L}_e}||\hat{\mathcal{U}_e} \cap \mathcal{U}_p|. \tag{4.9}$$

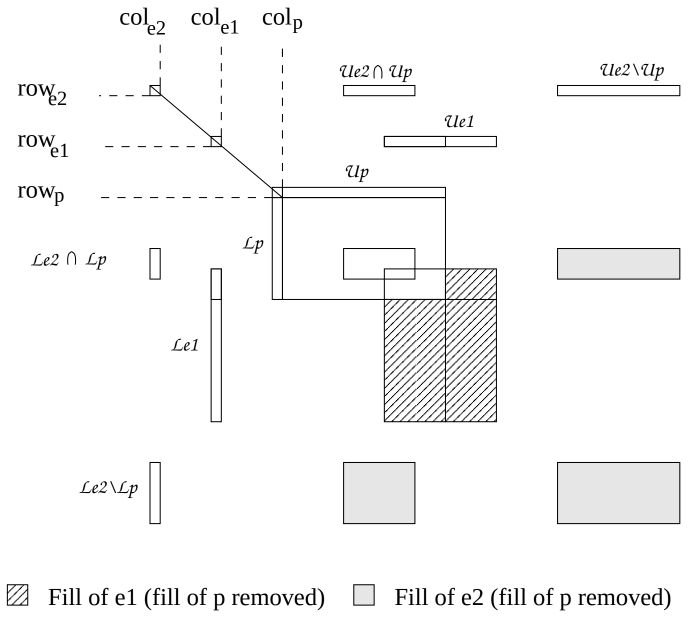


FIG. 4.5. AMFI *areas in the expanded matrix.*

Property 4.5 proves that one can in fact subtract area $AS(i,j)$, instead of $S(i,j)$, from equation (4.8), to obtain a more accurate upper bound of the fill metric.

PROPERTY 4.5. *$amd_r(i,j)amd_c(i,j) - |\hat{\mathcal{U}_p}||\hat{\mathcal{L}_p}| - AS(i,j)$ 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 4.5 is that, during the computation of the approximate degree, the submatrix is expanded in such a way that the intersections between all $\tilde{\mathcal{U}_e}$ and between all $\tilde{\mathcal{L}_e}$ 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 (see Figure 4.5). 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 4.5 is given in [32].

We now explain that, although computing $AS(i,j)$ is not trivial it is however not costly. To compute the area AS, we need to evaluate $\hat{\mathcal{U}_e}$, $\hat{\mathcal{L}_e}$, $\tilde{\mathcal{U}_e}$ and $\tilde{\mathcal{L}_e}$ (see equation (4.9)). It appears to be difficult to compute these quantities directly. Indeed only the quantities $|\mathcal{U}_e|$, $|\mathcal{L}_e|$, $|\mathcal{U}_e \setminus \mathcal{U}_p|$, $|\mathcal{L}_e \setminus \mathcal{L}_p|$, $|\mathcal{U}_e \cap \mathcal{U}_p|$ and $|\mathcal{L}_e \cap \mathcal{L}_p|$ are known thanks to AMD like computations. For each entry (i,j) involved in the metric update, two quantities are computed: an approximation of $AS(i,j)$, $area_{ij}$, such that

$$area_{ij} = \sum_{e \in \mathcal{F}} (|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|), \tag{4.10}$$

and local correction terms, cor_loc_{ij}, so that

$$AS(i,j) = area_{ij} - cor\_loc_{ij}. \tag{4.11}$$

Computing $area_{ij}$ only involves already known quantities. To compute the local correction terms cor_loc_{ij} we have to take into account for each $e \in \mathcal{F}$ all possible cases: $e \in \mathcal{R}_i \setminus \mathcal{C}_j$, $e \in \mathcal{C}_j \setminus \mathcal{R}_i$, and $e \in \mathcal{C}_j \cap \mathcal{R}_i$.

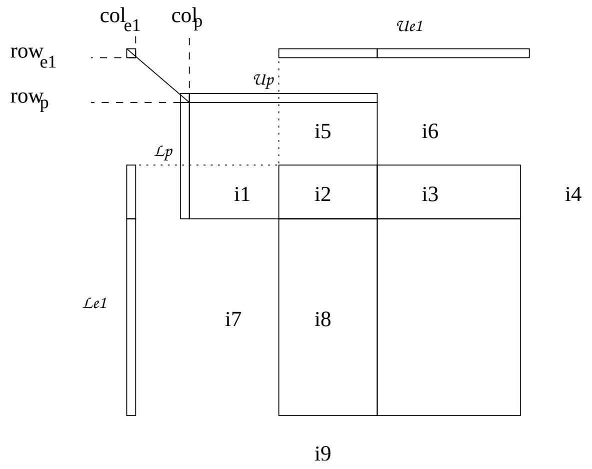


FIG. 4.6. AMFI*: different cases of local symmetrization.* $i1, i2, i3, i4, i5, i6$ *are cases encountered in loop 1,* $i7, i8, i9$ *are cases encountered in loop 2.*

Algorithm 4.2 explains how to compute both $area_{ij}$ and cor_loc_{ij} from which the AMFI metric can be deduced. To help understand how the local correction terms are computed, we indicate in Figure 4.6 the areas corresponding to different values of the local correction. In the following, we focus on the local correction terms computed during *Loop 1* of the algorithm. For $i \in \mathcal{L}_p$ (*Loop 1*), we note that, for each element $e \in \mathcal{R}_i$, $|\mathcal{U}_e \setminus \mathcal{U}_p|$ must be added to the correction term since it was taken into account by mistake in $|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e|$. In Figure 4.6, this situation occurs to cases i1, i2, i3 and i4 but not to cases i5 and i6 for which $|\mathcal{U}_e \setminus \mathcal{U}_p|$ is not added at line (1) of the algorithm since $e1 \notin \mathcal{R}_{i5}$ and $e1 \notin \mathcal{R}_{i6}$. We then see at line (2) of the algorithm that, since $j \in \mathcal{U}_p$, $|\mathcal{L}_e \setminus \mathcal{L}_p|$ must be added to the local correction (case i2). For case i3 at line [3] of the algorithm, the complete column structure $|\mathcal{L}_e|$ has been taken into account in $area_{ij}$ and must be added to the correction term. Furthermore, since the entry (i, j) had already been counted in $|\mathcal{U}_e \setminus \mathcal{U}_p|$, one should remove it from the correction term at line [3] of the algorithm. For case i5 (i6) in Figure 4.6, we add $|\mathcal{L}_e \setminus \mathcal{L}_p|$ ($|\mathcal{L}_e|$) that were counted by mistake in term $|\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$ ($|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e|$) of $area_{ij}$.

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

$$rowscale = \frac{\overline{amd}_r(i,j)}{amd_r(i,j)} \text{ and } colscale = \frac{\overline{amd}_c(i,j)}{amd_c(i,j)}.$$

If one systematically scales the area AS by $rowscale \times colscale$, then we ensure a positive metric and avoid tie-breaking problems due to metrics equal to 0. Our final

Algorithm 4.2 Computation of areas and local corrections for `AMFI` metric.

```
  for $i \in \mathcal{L}_p$ do /* Loop 1 */
    $area\_save_i = \sum_{e\in\mathcal{R}_i}(|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|)$
1   $cor\_loc\_save_i = \sum_{e\in\mathcal{R}_i}|\mathcal{U}_e \setminus \mathcal{U}_p|$ /* cases $i1, i2, i3$ and $i4$ */
    for each $j$ such that $(i,j) \in \mathbf{C}$ do
      $area_{ij} = area\_save_i$, $cor\_loc_{ij} = cor\_loc\_save_i$
      for each $e \in \mathcal{C}_j$ do
        if $e \in \mathcal{R}_i$ then /* $e$ already visited, its fill-in already counted in $area_{ij}$ */
          if $j \in \mathcal{U}_p$ then
2           $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e \setminus \mathcal{L}_p|$ /* case i2 */
          else
3           $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e| - 1$ /* case i3 */
          end if
        else /* $e$ not already visited, we need to count its corresponding area */
          $area_{ij} = area_{ij} + |\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$
          if $j \in \mathcal{U}_p$ then
4           $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e \setminus \mathcal{L}_p|$ /* case i5 */
          else
5           $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{L}_e|$ /* case i6 */
          end if
        end if
      end for
    end for
  end for
  for each $j \in \mathcal{U}_p$ do /* Loop 2 */
    $area\_save_j = \sum_{e\in\mathcal{C}_j}(|\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|)$
    $cor\_loc\_save_j = \sum_{e\in\mathcal{C}_j}|\mathcal{L}_e \setminus \mathcal{L}_p|$ /* used for cases $i8$ and $i9$ */
    for each row $i$ such that $(i,j) \in \mathbf{C}$ and $i \notin \mathcal{L}_p$ do
      $area_{ij} = area\_save_j$; $cor\_loc_{ij} = cor\_loc\_save_j$
      for each $e \in \mathcal{R}_i$ do
        if $e \notin \mathcal{C}_j$ then
          $area_{ij} = area_{ij} + |\mathcal{U}_e \setminus \mathcal{U}_p||\mathcal{L}_e| + |\mathcal{L}_e \setminus \mathcal{L}_p||\mathcal{U}_e \cap \mathcal{U}_p|$
          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{U}_e|$ /* case i7 */
        else
          $cor\_loc_{ij} = cor\_loc_{ij} + |\mathcal{U}_e| - 1$ /* case i8 */
        end if
      end for
    end for
  end for
```

`AMFI` metric is then defined as follows:

$$(4.12)\quad metric^{(k+1)}(i,j) = \min\begin{cases} \overline{amd}_r(i,j)\overline{amd}_c(i,j) - |\mathcal{U}_p \setminus j||\mathcal{L}_p \setminus i| \\ \quad - rowscale \times colscale \times \underline{AS}(i,j) \\ metric^{(k)}(i,j) \quad +|\mathcal{U}_p \setminus j| \times \overline{amd}_r(i,j) \\ \qquad\qquad\qquad +|\mathcal{L}_p \setminus i| \times \overline{amd}_c(i,j) \\ \qquad\qquad\qquad -2 \times |\mathcal{U}_p \setminus j| \times |\mathcal{L}_p \setminus i| \end{cases}$$

4.4.4. Complexity. First note that the complexity of the update of all the proposed metrics is comparable since the dominant cost associated with the computation of the approximate row and column degrees is shared by all metric calculations. We will thus focus here on the computation of the approximate Markowitz cost and more particularly on the computation of amd_c as defined by equation (4.2). The quantity $|\mathcal{A}_{*j} \setminus \mathcal{L}_p|$ is computed only once for all the entries in column j belonging to $\mathbf{C}$. The computation time is thus at each step bounded by $\mathcal{O}(\sum_j |\mathcal{A}_{*j}|) = \mathcal{O}(|E|)$. For each row in $\mathcal{L}_p$, the computational cost of the third term of Equation (4.2) is $\mathcal{O}(|\mathcal{R}_i|)$. For each row in $\mathcal{L}_p$ and each column j such that (i,j)

belongs to $\mathbf{C}$, the computational cost of the fourth term of equation (4.2) is $\mathcal{O}(|\mathcal{C}_j|)$. Thus the total complexity of amd_c for all the (i,j) entries in $(\mathcal{L}_p \times V_c) \cap \mathcal{P}attern(\mathbf{C})$ is

$$\mathcal{O}\left(\sum_{i\in\mathcal{L}_p} |\mathcal{R}_i| + \sum_{i\in\mathcal{L}_p}\sum_{j\in R_i} |\mathcal{C}_j|\right). \tag{4.13}$$

Let us define $n_1, \ldots, n_{|\mathcal{L}_p|}$ by induction: $n_1 = arg\max|\mathcal{C}_j|$ and for $i > 1$, $n_i = arg\max_{j\neq n_{i-1},\ldots,n_1} |\mathcal{C}_j|$. At each step p of the symbolic factorization, Let us define k^p as the largest row or column length. $k^p = \max(\max_{i\in V_r} |R_i^p|, \max_{j\in V_c} |C_j^p|)$. The double sum of equation (4.13) thus contains at most $k^p|\mathcal{L}_p|$ terms where k^p is defined as the size of the largest row/column at step p. Note that at most k^p terms can be equal to $\mathcal{C}_{n_i}$ for all $i \in [1, |\mathcal{L}_p|]$. Thus, we have the following inequality: $\sum_{i\in\mathcal{L}_p}\sum_{j\in R_i} |\mathcal{C}_j| \leq k^p \times \sum_{1\leq i\leq|\mathcal{L}_p|} |\mathcal{C}_{n_i}|$. Then we have

$$\sum_{i\in\mathcal{L}_p} (|\mathcal{R}_i| + \sum_{j\in R_i} |\mathcal{C}_j|) \leq k^p \times \left(\sum_{i\in\mathcal{L}_p} |\mathcal{R}_i| + \sum_{1\leq i\leq|\mathcal{L}_p|} |\mathcal{C}_{n_i}|\right).$$

Finally, thanks to the in-place property of our algorithm to update the bipartite quotient graph we obtain

$$\sum_{i\in\mathcal{L}_p} |\mathcal{R}_i| + \sum_{1\leq i\leq|\mathcal{L}_p|} |\mathcal{C}_{n_i}| \leq |E|.$$

The same bounds can be obtained for amd_r so that the total cost for updating the metric is $\mathcal{O}(k^p|E|)$ per elimination step. The total time complexity is $\mathcal{O}(\sum_{1<p\leq n} k^p|E|)$. If k^p is bounded, the complexity becomes $\mathcal{O}(n|E|)$ which is the same complexity as the `AMD` algorithm [26].

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.

4.5.1. Adaptation of CMLS main scheme. 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, 4, 16, 21] 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 [20]: we define *indistinguishable row variables* (resp. *indistinguishable column variables*) as row variables (resp. columns variables) which have the same adjacency in $\mathcal{G}$. To limit the cost of supervariable detection, two hash functions (see for example [7]) are then used for each row and column direction.

If i and j are two indistinguishable row variables, they are replaced in $\mathcal{G}$ by a *row supervariable* containing both i and j, labeled by its *principal row variable* (i, say) [15, 16, 17]. 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 pivot $p_1 = (r_1, c_1)$ is eliminated, $p_2 = (r_2, c_2)$ can be eliminated in $\mathcal{G}$ without causing extra fill-in. This process, commonly referred to as *mass elimination* [23], creates a new (super)element $e = (\mathbf{r}, \mathbf{c})$ in the quotient graph. In the following, we comment on the algorithmic modifications due to the introduction of supervariables.

Let $p = (\mathbf{r}, \mathbf{c})$ be the current pivot. The first modification of the algorithm concerns the introduction of a scaling of the structural metrics as defined either by equation (4.5) or by equation (4.12). The structural metric of an entry (i, 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 he elimination process which is performed in the three main steps. During the first step, the scaled metric is use to select a pivot in $\mathbf{C}$. During the second step, we retrieve its associated row $\mathbf{r}$ and column $\mathbf{c}$ supervariable in $\mathbf{A}$. During the third step, we eliminate "as many as possible" variables belonging to $(\mathbf{r} \times \mathbf{c}) \cap \mathbf{C}$. 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 in the intersection of the pattern of $\mathbf{C}$ and the supervariables need also be considered. The same modified three steps are also applied to the mass elimination process. 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 $\mathcal{G}$.

The final modification concerns the update of the structural metric that will be fully described in Section 4.5.2. Note that during the metric update, we also mark candidates for mass elimination.

4.5.2. Revisiting computation of the structural metrics. When using supervariables and after the elimination of a pivot p, the approximate external row and column degrees as defined by equations (4.1) and (4.2) become:

$$\begin{aligned} amd_r(i,j) = \quad & |\mathcal{A}_{\mathbf{i}*} \setminus \mathcal{U}_p| + |\mathcal{U}_p \setminus \{\mathbf{j}\}| + \textstyle\sum_{e \in \mathcal{R}_{\mathbf{i}} \cup \mathcal{C}_{\mathbf{j}}} (|\mathcal{U}_e \setminus \mathcal{U}_p|) - \alpha_{\mathbf{j}}\ |\mathbf{j}|, \\ & \text{with } \alpha_{\mathbf{j}} = \max(|\mathcal{C}_{\mathbf{j}}|, 1) \text{ if } \mathbf{j} \notin \mathcal{U}_p \text{ else } \alpha_{\mathbf{j}} = 0. \end{aligned} \tag{4.14}$$

$$\begin{aligned} amd_c(i,j) = \quad & |\mathcal{A}_{*\mathbf{j}} \setminus \mathcal{L}_p| + |\mathcal{L}_p \setminus \{\mathbf{i}\}| + \textstyle\sum_{e \in \mathcal{R}_{\mathbf{i}} \cup \mathcal{C}_{\mathbf{j}}} (|\mathcal{L}_e \setminus \mathcal{L}_p|) - \beta_{\mathbf{i}}\ |\mathbf{i}|, \\ & \text{with } \beta_{\mathbf{i}} = \max(|\mathcal{R}_{\mathbf{i}}|, 1) \text{ if } \mathbf{i} \notin \mathcal{L}_p \text{ else } \beta_{\mathbf{i}} = 0. \end{aligned} \tag{4.15}$$

In Section 4.4.3, we have shown that to evaluate the AMFI metric of an entry (i, j) we need to compute the area $AS(i, j)$ of equation (4.12) as the sum of two terms $area_{ij}$ and cor_loc_{ij} (see equation (4.11)). The $area_{ij}$ term (see equation (4.10)) is related to the areas of all the bi-cliques of the already eliminated elements so that its computation does not depend on the use of supervariables. The computation of the local correction terms cor_loc_{ij} must however be revisited. We split this correction term into two parts. Each part, $row\_cor\_loc_{ij}$ and $col\_cor\_loc_{ij}$, refers respectively to entries in the row supervariable and to entries in the column supervariable so that the new equation for defining the area AS is :

$$AS(i,j) = area_{ij} - row\_cor\_loc_{ij}\ |\mathbf{i}| - col\_cor\_loc_{ij}|\mathbf{j}|. \tag{4.16}$$

For an entry $(i,j) \in \mathbf{C}$, the local corrections for the different cases of Figure 4.6 are then:

- Case $i1$: $row\_cor\_loc_{ij} = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc_{ij} = 0$,
- Case $i2$: $row\_cor\_loc_{ij} = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i3$: $row\_cor\_loc_{ij} = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e| - |\mathbf{i}|$,
- Case $i4$: $row\_cor\_loc_{ij} = |\mathcal{U}_e \setminus \mathcal{U}_p|$ and $col\_cor\_loc_{ij} = 0$,
- Case $i5$: $row\_cor\_loc_{ij} = 0$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i6$: $row\_cor\_loc_{ij} = 0$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e|$,
- Case $i7$: $row\_cor\_loc_{ij} = |\mathcal{U}_e|$ and $col\_cor\_loc_{ij} = 0$,
- Case $i8$: $row\_cor\_loc_{ij} = |\mathcal{U}_e| - |\mathbf{j}|$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e \setminus \mathcal{L}_p|$,
- Case $i9$: $row\_cor\_loc_{ij} = 0$ and $col\_cor\_loc_{ij} = |\mathcal{L}_e \setminus \mathcal{L}_p|$.

It is then straightforward to accumulate these corrections by adapting Algorithm 4.2.

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 [13, 14] because it is the most robust in-place local heuristic (better than AMD, see [4]) 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 **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 some 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 **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 **C**.

To represent each class of solver techniques, we consider the multifrontal code MA41_UNS [2, 5] 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 [3, 5, 10, 25] 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, 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 first briefly 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 solver 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 the SuperLU_DIST context and focus in the case on the numerical effects.

5.1. Experimental environment.

5.1.1. Test matrices and computing environment. A representative set of 19 large unsymmetric matrices has been selected (see Table 5.1) from Tim Davis' collection [9]. 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 GOTO BLAS [24].

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:

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

In the remainder of this section, the symmetry of a matrix always refers to the structural symmetry once the MC64 permutation has been applied (see column *sym* of Table 5.1).

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

TABLE 5.1
Test matrices.

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.

5.1.2. CMLS default parameters. The initialization of $\mathbf{C}$ is done using 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 $4n$. We then drop the entries that are smaller than 0.1 in magnitude and the entries that have too large structural metrics. While dropping, we still maintain the property $\mathcal{M} \subset \mathcal{P}attern(\mathbf{C})$. On 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 Improved Approximate Minimum Fill metric `AMFI` of Section 4.4.3 since it is the most efficient metric for both `CMLS` and `DMLS` orderings.

5.2. Preliminary remarks about diagonal constraint matrices. 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 (see [32] for detailed results) that `CMLS` ordering tends to produce sparser factors even if `DMLS` uses two-way variable elimination which leads (as explained in Section 4.4.1) to more accurate structural metrics. 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. It has more chance to better preserve the sparsity of reducible matrices. 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`. Note that on the other matrices the number of detected blocks does not depend on the ordering and thus the algorithm used for variable elimination will not influence much the results on our set of test matrices.
- `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.
- In the `CMLS` implementation, we use *rowscale* and *colscale* coefficients (see end of Section 4.4.3) to reduce the amount of tie-breaking between variables that would have a negative metric (reset to 0) with `DMLS`.

We should stress that most of 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 the best implementation of `DMLS` should use 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 0% to 43%, with gains very much comparable for both the estimated and the real size of the factors.

Matrix	Estimated size of factors CMLS	DMLS	Real size of factors CMLS	DMLS	Ratio: actual/predicted CMLS	DMLS
av41092	6610	9359	7237	9965	1.09	1.06
g7jac200sc	28217	36345	29035	37001	1.02	1.01
g7jac180sc	21561	31797	22222	32391	1.03	1.01
jan99jac120sc	3793	3774	4190	4221	1.10	1.11
jan99jac100sc	3172	3119	3505	3496	1.10	1.12
lhr34c	6612	6833	7522	7821	1.13	1.14
lhr71c	13958	15274	15860	17450	1.13	1.14
mark3jac120sc	13079	13433	13658	14140	1.04	1.05
mark3jac140sc	15127	15621	15790	16452	1.04	1.05
bayer01	1374	2410	1633	2737	1.18	1.13
sinc18	31314	37971	32382	39878	1.03	1.05
sinc15	15453	16974	16093	17343	1.04	1.02
Zhao2	12819	13937	13453	14689	1.04	1.05
mult_dcop_03	780	865	959	1022	1.22	1.18
twotone	7469	8452	8377	9099	1.12	1.07
onetone1	2827	3174	3125	3484	1.10	1.09
torso1	30605	30639	31491	31544	1.02	1.02
bbmat	37942	43382	38549	43915	1.01	1.01
shermanACb	365	396	465	487	1.27	1.22
Mean CMLS/DMLS	0.87		0.88			

TABLE 5.2
MA41_UNS *size of the factors and analysis reliability. Each number for the factor size is in thousands.*

As expected the fact that more flexibility has been offered to select off-diagonal pivots in the constraint matrix help CMLS at preserving the sparsity of the factors. Doing so we have however authorized CMLS to select pivots that do not belong to the maximum weighted matching. Since, a structural metric has then been used by CMLS algorithm to select the pivots, it is thus critical to evaluate the numerical quality of our sequence of pivots 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 factor. 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 (both delayed pivots and amalgamation contributes to increasing this ratio). Note that from a software point of view it is also critical for the estimation to reflect the reality. We see in Table 5.2 that the growth of the size of the factors is reasonable. Note that the fill-in due to the amalgamation process can be predicted and so if node amalgamation were included in the estimation, the growth of the size of the factors would be even smaller. Furthermore, it is even more interesting to observe that the growth obtained with CMLS is comparable to that obtained with DMLS. This shows that limiting the pivot choice to the maximum weighted matching (as done in DMLS) and considering numerical information in the constraint matrix (hybrid strategies to select pivots) are not critical in the context of a partial pivoting code.

5.3.2. Run-time and memory usage. In this section, we are concerned with 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.

Matrix	Number of operations CMLS	DMLS	Memory needed CMLS	DMLS
av41092	1876.7	3809.2	7610	10066
g7jac200sc	27330.2	44379.8	30328	40543
g7jac180sc	17005.9	38434.8	23153	34577
jan99jac120sc	1162.5	1141.8	4318	4303
jan99jac100sc	1039.0	938.3	3577	3554
lhr34c	1609.4	1812.7	7766	8389
lhr71c	3567.8	4981.5	16139	18052
mark3jac120sc	6786.6	7142.8	14036	15150
mark3jac140sc	7661.2	8135.0	16273	17153
bayer01	58.9	155.4	1635	2743
sinc18	55501.4	77367.7	35369	46735
sinc15	19521.4	22388.5	20092	19915
Zhao2	8023.3	9250.6	14400	15354
mult_dcop_03	85.7	114.6	1011	1073
twotone	5142.9	5591.1	9296	9282
onetone1	1023.2	1250.3	3532	3791
torso1	24417.6	25068.9	34364	35256
bbmat	30455.0	40766.0	38941	44448
shermanACb	22.3	27.7	490	536
Mean CMLS/DMLS	0.79		0.88	

TABLE 5.3
MA41_UNS *number of operations (in millions) and memory used (in thousands of reals).*

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 12%. The reduction in the number of operation is even more important (mean value of 21%) and will contribute to the reduction in the factorization time.

Matrix	ordering time CMLS	DMLS	factorization time CMLS	DMLS	solution time CMLS	DMLS
av41092	3.79	3.62	1.57	2.52	0.69	0.70
g7jac200sc	22.25	9.46	16.17	23.95	1.50	2.36
g7jac180sc	16.34	7.42	12.14	19.95	1.51	2.06
jan99jac120sc	6.07	2.59	1.68	1.48	0.35	0.31
jan99jac100sc	4.78	1.92	1.616	1.17	0.32	0.28
lhr34c	3.56	3.22	11.85	23.11	0.84	0.93
lhr71c	8.28	7.87	19.87	39.51	1.88	2.31
mark3jac120sc	7.85	3.39	4.53	3.86	1.08	1.10
mark3jac140sc	9.56	4.20	5.16	4.54	1.09	1.42
bayer01	1.58	1.14	0.32	0.39	0.59	0.49
sinc18	16.54	18.03	22.99	26.55	0.66	0.91
sinc15	7.73	3.58	8.77	10.55	0.35	0.38
Zhao2	2.15	0.90	4.47	4.97	0.46	0.83
mult_dcop_03	0.61	0.47	0.24	0.27	0.24	0.27
twotone	3.58	2.09	4.69	6.34	1.30	1.32
onetone1	1.02	0.53	0.86	1.32	0.25	0.26
torso1	14.50	68.27	11.68	11.61	2.45	2.40
bbmat	44.74	16.50	31.60	37.67	1.57	1.37
shermanACb	0.26	0.14	0.09	0.08	0.09	0.10
Mean	1.77		0.86		0.91	

TABLE 5.4
MA41_UNS *ordering, factorization and solution time (in seconds).*

Table 5.4 then compares the time of the three main steps of the resolution. Note that the ordering time of both orderings depends on two opposite effects that are difficult to detect. The better we preserve sparsity, the smaller might be the quotient graph, and the faster we can process it. The better we preserve sparsity, the fewer the elements are absorbed, the fewer the supervariables are detected, and 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 $\mathbf{C}$ (see Section 4.4.4). Considering that the size of the $\mathbf{C}^0$ is typically between $2n$ and $3n$, we see in Table 5.4 that `CMLS` is quite competitive with respect `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. 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 avoids such computation. It sometimes decreases significantly the ordering time (see for example torso1 matrix). Furthermore, supervariables have been generalized in the context to `CMLS` ordering resulting in separated row and column supervariables. This feature helps `CMLS` exploit in a more efficient way the unsymmetric structure of the matrix.

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 decrease in the factorization time. The gains in factorization time are slightly smaller than the gains in the number of operations (the average decrease in the number of operations is around 21%). This is because sparser factors often leads to smaller full blocks on 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 1.33 GFlops with the `DMLS` ordering whereas it is around 1.23 GFlops with the `CMLS` ordering. The largest gains in time are then obtained for matrices with small flop rate. For example, for the lhr* matrices the average flop rate is around 100 MFlops, and `MA41_UNS` is two times faster with `CMLS` than with `DMLS`.

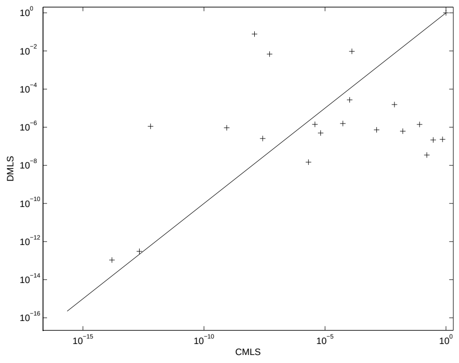
5.4. Impact of the hybrid strategies on `SuperLU_DIST`. Because of the static pivoting strategy used during factorization, `SuperLU_DIST` is expected to be numerically more sensitive than `MA41_UNS` to the use of hybrid strategies to select pivots during the `CMLS` ordering. We thus focus in this section, on the analysis of the numerical behaviour of `SuperLU_DIST`. It has been observed in [3] that, because of static pivoting, iterative refinement may be required to obtain an accurate solution. We thus analyse in this section the component-wise backward error of the solution [6] 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 thus very much 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 $\mathbf{C}$. The relative threshold was set to 0.01 in all our experiments.

We added to our test set four matrices (see Table 5.5) on which we have observed in [3] that `SuperLU_DIST` need iterative refinement to improve the accuracy of the solution.

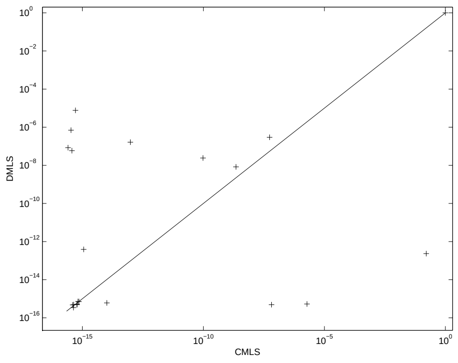
Figure 5.1 compares `CMLS` and `DMLS` component-wise backward error during the iterative refinement (results after 0, 2, 3, and 4 steps). From the results it is clear that using the `CMLS` ordering clearly improves the numerical behavior of `SuperLU_DIST`.

Group/Matrix	n	nnz	Origin
MIXING-TANK	29957	1995041	fluid flow (PARASOL, Polyflow S.A.)
INV-EXTRUSION-1	30412	1793881	fluid flow (PARASOL, Polyflow S.A.)
fidapm11	22294	623554	CFD (SPARSKIT2 collection)
cavity16	4562	138187	Finite element modeling (SPARSKIT2 collection)

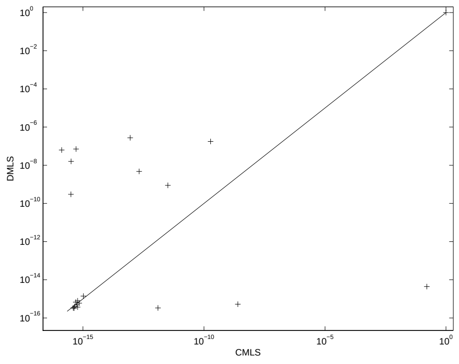
TABLE 5.5
Test matrices.



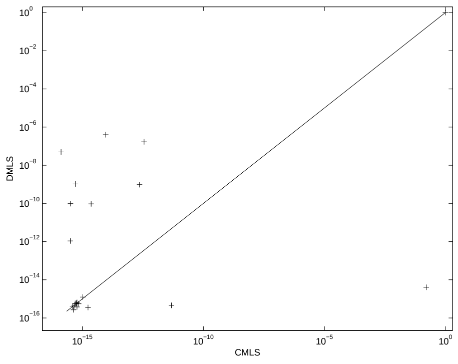
(a) Component-wise backward, step 0.



(b) Component-wise backward, step 2.



(c) Component-wise backward, step 3.



(d) Component-wise backward, step 4.

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

There are still two matrices (av41092 and Zhao2) for which, after both CMLS and DMLS orderings, iterative refinement does not converge to an accurate solution. The torso1 matrix is the only one on which a DMLS approach succeeds whereas a CMLS approach fails. There are seven matrices (sinc18, lhr34c, lhr71c, mult_dcop3, MIXING-TANK, fidapm11 and cavity16) on which the backward error of SuperLU_DIST combined with DMLS remains larger than 10^{-10} whereas SuperLU_DIST combined with CMLS converges in less than 4 iterations. Note finally that, with CMLS, on all problems (except torso1, av41092 and Zhao2), three steps of iterative refinement are sufficient to obtain a backward error smaller than 10^{-8}.

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 properties of the algorithm are summarized as follows:

- Significant gains in terms of fill-in (13%) and flops (21%) have been obtained with the structural strategy to select the pivots.
- Using a 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.
- Although the complexity of the ordering is higher with CMLS than with DMLS, we have shown that CMLS benefits from algorithmic improvements (some of which could even be implemented in DMLS).

The last property emphasizes that DMLS could also benefit from the algorithms developed in the more general and complex framework used for the CMLS algorithm. Both our approach to scale metrics in order to achieve better tie-breaking and 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 simpler or easier to parallelize version of the preprocessing phase.

Furthermore, the constraint matrix **C** contains information that can be seen as an incomplete factorization. We intend to use it as a preconditioner and to compare its quality and cost with existing incomplete **LU** factorizations.

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*, Int. J. of Supercomputer Applics., 3 (1989), pp. 41–59.

[3] 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.

[4] 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.

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

[6] 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.

[7] C. Ashcraft, *Compressed graphs and the minimum degree algorithm*, SIAM J. Sci. Comput., 16 (1995), pp. 1404–1411.

[8] 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.

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

[10] T. A. Davis, *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, Tech. Rep. TR-00-006, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 2003. To appear in TOMS.

[11] 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.

[12] 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.

[13] 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.

[14] ———, *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.

[15] 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.

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

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

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

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

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

[21] A. George and J. W. H. Liu, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Trans. Math. Softw., 6 (1980), pp. 337–358.

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

[23] 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.

[24] K. Goto and R. Geijn, *On reducing tlb misses in matrix multiplication*, 2002.

[25] 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.

[26] P. Heggernes, S. Eisenstat, G. Kumfert, and A. Pothen, *The computational complexity of the minimum degree algorithm*, in Proceedings of the Norwegian Conference on Computer Science NIK, 2002.

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