DECOMPOSITION METHODS FOR SOLVING SPARSE NONLINEAR SYSTEMS OF EQUATIONS

ALI BAHAREV, HERMANN SCHICHL, ARNOLD NEUMAIER

Abstract.
We survey decomposition methods that aim at reducing the computation time of solving a given sparse nonlinear system of equations. After the decomposition, a sequence of small subproblems determined from a suitable ordering of the variables and equations are solved. The ultimate objective is to find that ordering that results in the most savings with respect to the overall solution time, including the time to determine the ordering. Since this objective is solver and machine dependent, and too complicated to handle rigorously, compromises have to be made. In the context of nonlinear systems of equations, a popular objective is to maximize the number of variables eliminated by solving univariate equations. This objective makes the problem fundamentally different from fill-reducing orderings. Throughout this paper, the decomposition methods are treated in terms of sparse matrix orderings: The numerical methods for solving the decomposed systems are not discussed.

Key words. design structure matrix, diakoptics, minimum degree ordering, sparse matrix ordering, tearing

1. Introduction. One of the most popular decomposition methods for solving nonlinear systems of equations is called tearing. Three equivalent formal definitions of optimal tearing are given: Section 2 defines it in terms of sparse matrices, Section 3 defines it with the aid of sparse bipartite graphs, and Section 4 gives an integer linear programming (ILP) formulation. Problems closely related to tearing are discussed in Section 5. In particular, in Section 5.1 we relate tearing, as defined in the chemical engineering literature, to the definitions given in Sections 2–4.

The goal of tearing is to reduce the computation time needed to solve a given system of equations by exploiting its sparsity pattern. The exact running time depends on many factors beside the input problem, and this objective would be too difficult to handle rigorously. Therefore, in practice, simpler objective functions are used, and it is assumed that minimizing the chosen objective function also minimizes the running time (at least approximately). The choice of the objective function is a matter of judgment and highly context dependent. In the context of nonlinear systems of equations, a popular objective is to maximize the number of variables eliminated by solving univariate equations. This objective is used in the present paper too. However, this choice is a compromise, and the issues of this objective will be discussed in Section 6.

It is extremely difficult to categorize variations and improvements of tearing, since they often overlap and do not fit cleanly into a single category. Nevertheless, we try to categorize them in Sections 7–10.

Certain variants of tearing are discussed in Section 7 that allow small solvable subsystems other than univariate equations. As we will discuss in Section 7.1, a typical implementation starts with the so-called block lower triangular decomposition, and tearing is only applied to the obtained diagonal blocks. We give a simple example showing that this approach can lead to suboptimal results. The gap between the cost of the ordering obtained with such an implementation and the cost of the optimal ordering can be proportional to the problem size. This is unacceptable for the kind of applications we care about (for example distillation columns); Sections 7.2–7.4 describe variants of tearing that can produce appropriate orderings in such cases.

Section 8 presents ways to improve heuristic methods for tearing, while Section 9 gives examples for improving exact tearing algorithms. Section 10 briefly describes

context dependent alternative objective functions. The paper ends with Section 11 listing various practical applications of tearing.

The sibling paper [9] of the present paper introduces two exact algorithms for optimal tearing: (i) an algorithm based on integer programming, and (ii) a custom branch and bound algorithm. It also proposes a simple algorithm for automatically identifying numerically safe eliminations (that avoids division by zero for example).

2. Tearing of sparse matrices.

2.1. Tearing. **Tearing** (cf. [38, 39, 87]) is the representation of a sparse system of nonlinear equations

$$f(x) = 0, \text{ where } f: \mathbb{R}^n \mapsto \mathbb{R}^m, \tag{1}$$

in a permuted form where most of the variables can be computed sequentially once a small auxiliary system has been solved. More specifically, given permutation matrices P and Q such that after the transformation

$$\begin{bmatrix} g \\ h \end{bmatrix} = Pf, \qquad \begin{bmatrix} y \\ z \end{bmatrix} = Qx, \tag{2}$$

$g_i(y, z) = 0$ can be rewritten in the equivalent explicit form

$$y_i = \tilde{g}_i(y_{1:i-1}, z) \tag{3}$$

using appropriate symbolic transformations. Equation (3) implies that the sparsity pattern of the Jacobian of Pf is

$$J = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \text{ where } A \text{ is lower triangular}, \tag{4}$$

J is therefore **bordered lower triangular**. Hereafter, we will refer to a particular choice of P, Q, g, h, y, and z satisfying equations (3) and (4) as an **ordering**. Given an ordering, the system of equations $f(x) = 0$ can be written as

$$\begin{aligned} g(y, z) &= 0 \\ h(y, z) &= 0. \end{aligned} \tag{5}$$

The requirement (3) that $g_i(y, z) = 0$ can be made explicit in y_i essentially means $y = \bar{g}(z)$. Substituting y into h yields $h(\bar{g}(z), z) = 0$ or

$$H(z) = 0. \tag{6}$$

That is, the original nonlinear system of equations $f(x) = 0$ is reduced to the (usually much) smaller system $H(z) = 0$.

Notes. If (3) cannot be made explicit in y_i, one can still attempt to solve for y_i numerically, see for example `local_iteration_in_tearing` in [88]. This can help to reduce the dimension of z, as the variable y_i otherwise would have to be in z.

It is assumed throughout this paper that each equation involves at least one variable. (For example, the equation $0 = 0$, although true, is not considered a valid input for the proposed algorithms.) It is also assumed that each variable appears in at least one equation, that is, free variables are not allowed. A preprocessor can easily remove such equations and variables from the input.

2.2. Optimal tearing. Optimal tearing is the task of finding an ordering that minimizes the border width

$$d := \dim z \tag{7}$$

of J. This objective is a popular choice [16, Sec. 8.4] and often results in a significant speed up, although it does not guarantee any savings in computation time in the general case. We will discuss the issues related to this objective in Section 6.

When seeking the solutions of (1) with a numerical solver, instead of posing (1) to the solver, we pose the smaller system (6). Once we have a solution z to (6), we recover the solution x to the original system (1) using (2) and (3). There are exactly two types of variables: (i) the **eliminated variables** y get their value through assignment according to (3), and (ii) the **guessed variables** z, that are to be determined by the solver. We call them guessed variables because in some sense the solver has to "guess" the correct values of these variables such that (6) is satisfied. Also, solvers usually require that the user provides an initial guess for the variables, for z in this case. By minimizing d, we minimize the guesswork; only the guessed variables are counted in the objective.

3. Tearing of sparse bipartite graphs. The adjacency matrix associated with system (1) is represented as a bipartite graph B. One of the disconnected node sets is denoted by R and corresponds to the equations (rows); the other node set is denoted by C and corresponds to the variables (columns). If the variable x_j appears in equation $f_i(x) = 0$, there is an edge between the respective two nodes of the bipartite graph; the two nodes are not connected otherwise. We assume that every node has at least one incident edge, otherwise the corresponding variable or equation can be ignored, reducing the problem size. An example is shown in Figure 1.

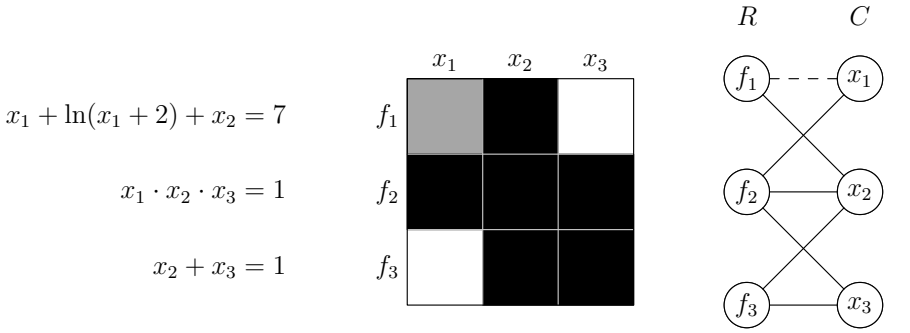


FIG. 1. *An example system of equations, the corresponding sparsity pattern of the Jacobian, and bipartite graph B. Note that f_1 cannot be made explicit in x_1; the corresponding entry is gray in the matrix, and the corresponding edge of the graph is dashed.*

A subset of the edges of B is labeled, let F denote this set. F can be an arbitrary subset of the edges, but the intention is to label those edges that, when brought into explicit form as in (3), are unique in y_i and numerically stable. The goal is to maximize the number of variables eliminated through such assignments. Identifying a suitable set F is discussed in [9].

On a high level of abstraction, we can think of tearing as follows.

1. We first orient the bipartite graph B associated with system (1): The edges of the graph are directed in a special way, determined by a matching algorithm.
2. The resulting directed graph is made acyclic by reversing as few of its edges as possible.
3. To find an optimal elimination order, the cardinality of the eliminated variables has to be maximized over all possible orientations of B .

This interpretation provides the basis of an integer programming-based approach, to be presented in the next sections. However, the above view of tearing, performing the individual steps sequentially, is only conceptual: Greedy heuristics exist that perform the orientation, edge reversing, and ordering steps simultaneously, see for example [23, 24, 116]. When such a greedy heuristic is used, the simultaneous approach can be fruitful: One can take into account the resulting cycles when orienting the bipartite graph. On the other hand, exact methods must consider all the possible orientations.

3.1. Matching. We introduce some terminology before giving further details of the individual steps mentioned above. Given an undirected graph $G = (V, E)$, a **matching** is a subset of edges $M \subseteq E$ such that for all nodes $v \in V$, at most one edge of M is incident on v. A node $v \in V$ is **matched** by the matching M if some edge in M is incident on v; otherwise, v is **unmatched**. A **perfect matching** is a matching in which every node is matched. A **maximum matching** is a matching of maximum cardinality, that is, a matching M such that for any matching M', we have $|M| \geq |M'|$. If a bipartite graph is viewed as a sparse matrix, a maximum matching corresponds to a permutation that places a maximum number of nonzero elements on the diagonal of the matrix. In this context, maximum matching is also referred to as **maximum transversal** or **maximum assignment**. In the earlier chemical engineering literature, matching is also referred to as **output set assignment**: We assign each eliminated variable ("output") to the equation used for eliminating that variable.

3.2. Orientation. Given an arbitrary bipartite matching M', we first remove those edges from M' that are not in F; the resulting subset $M = M' \cap F$ of edges is still a matching. Then, the edges of the bipartite graph B are directed such that the edges in M point towards the nodes in C, and all the other edges point towards the nodes in R; see Figure 2.

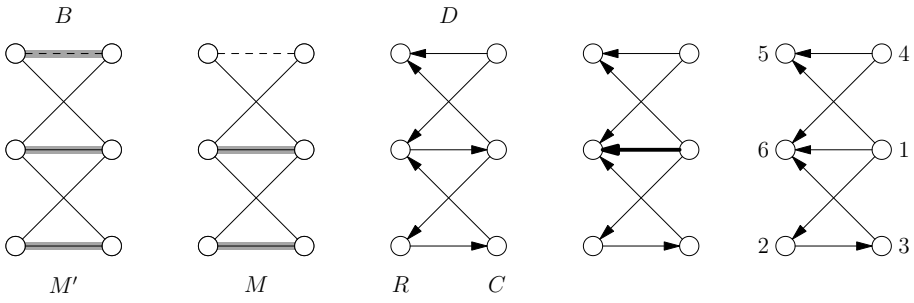


FIG. 2. *The steps of tearing: bipartite matching M' → matching M after considering feasible assignments → orientation → feedback edge set → a possible elimination order. Details are given in the text.*

3.3. Cycle structure. The way the bipartite graph B is oriented given the matching M has implications on the cycle structure of the resulting directed graph D. An unmatched node $r \in R$ must be a sink; an unmatched node $c \in C$ must be a

source. Neither sinks, nor sources can participate in any cycle, only matched nodes can. The only matched edge of a matched node n must participate in all the cycles that n is involved in; reversing the only matched edge of n therefore must destroy all the cycles passing through n and cannot create new cycles. Reversing an edge $(u, v) \in M$ of D has the same effect as removing (u, v) from M and re-orienting B; $M \setminus \{(u, v)\}$ is still a matching, u and v become a source and a sink accordingly.

In general, a **simple cycle** in a (directed or undirected and not necessarily bipartite) graph is a cycle with no repeating edges and no repeating nodes in the cycle. Two simple cycles are **distinct** if one is not a cyclic permutation of the other. Throughout this paper, whenever simple cycles are mentioned, distinct simple cycles are meant.

Let us narrow our attention to the subgraph H induced by the edges participating in a simple cycle of an undirected bipartite graph. There are exactly two orientations of H such that a directed cycle is obtained; both of these orientations stem from a perfect matching, or equivalently, a maximum cardinality matching in this case. Exactly half of the edges of H are included in a maximum cardinality matching, see Figure 3. (An undirected graph G is bipartite if and only if every cycle of G has even length [131, Sec. 2.1].) Therefore, given a bipartite graph B and a matching M, the

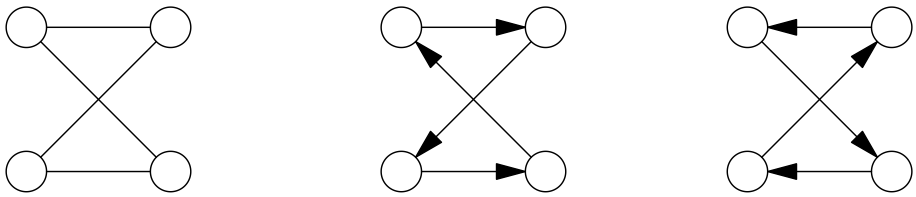


FIG. 3. *An undirected simple cycle and its two orientations that result in directed simple cycles.*

orientation of B according to M will result in a directed acyclic graph if and only if M does not involve a maximum cardinality matching on any subgraph induced by the nodes of a simple cycle.

3.4. The feedback edge set problem. The subset of edges in M that have to be reversed to make the resulting directed graph acyclic are called a feedback edge set. In this subsection, we define the minimum feedback edge set problem in the general case, where the input directed graph G is not necessarily bipartite.

A **feedback edge set** is a subset of edges containing at least one edge of every cycle in a (not necessarily bipartite) directed graph G. Removing the edges in the feedback edge set from G makes the remaining graph acyclic. The feedback edge set is often referred to as **feedback arc set** in the literature, however, we prefer the term feedback edge set.

Given a directed graph G and an integer parameter k, the **parameterized feedback edge set problem** is to either construct a feedback edge set of at most k edges for G, or to prove that no such edge set exists. This problem is called the feedback arc set problem (item 8) on the list of Richard M. Karp's 21 NP-complete problems [68].

Finding a feedback edge set of minimum cardinality is the **minimum feedback arc set problem**; we will refer to it as the **minimum feedback edge set problem** hereafter. The complementary problem to the minimum feedback edge set problem is the **maximum acyclic subgraph problem**. Unless $P = NP$, the minimum feedback edge set problem does not have a polynomial-time approximation scheme [67]. The minimum feedback edge set problem is **approximation resistant**: Conditioned on the Unique Games Conjecture (UGC) [69], for every $C > 0$, it is NP-hard to

find a C-approximation to the minimum feedback edge set problem, see Corollary 1.2. in [54]. Both heuristics for finding a feedback edge set of small cardinality and exact algorithms for finding a feedback edge set of minimum cardinality are discussed in [10].

The eliminated variables correspond to those nodes of the final directed acyclic graph that have an incoming edge; these edges were originally in the matching M as well. The objective of tearing is to find an orientation (or equivalently a matching) that maximizes the cardinality of the eliminated variables.

4. Tearing by integer linear programming. The following integer programming formulation is used in our implementation; any feasible solution to this integer program uniquely defines a bipartite matching M.

$$
\begin{aligned}
&\max_{y} \sum_{e\in F} y_e && \text{(find the maximum-cardinality matching)}\\
&\text{s.t.} \sum_{e\in E} u_{re} y_e \le 1 \quad \text{for each } r \in R, && \text{(each row is matched at most once)}\\
&\sum_{e\in E} v_{ce} y_e \le 1 \quad \text{for each } c \in C, && \text{(each column is matched at most once)}\\
&\sum_{e\in E} a_{se} y_e \le \frac{\ell_s}{2} - 1 \quad \text{for each } s \in S && \text{(cycles are not allowed).}
\end{aligned}
\tag{8}
$$

Here the binary variable y_e is 1 if edge e is in the matching M, and 0 otherwise; the set F is an arbitrary subset of edges of B, see Sec. 3; E, R, and C denote the index sets of the edges, the rows, and the columns, respectively; u_{re} is 1 if node r is incident to edge e, and 0 otherwise; similarly, v_{ce} is 1 if node c is incident to edge e, and 0 otherwise; S is the index set of those simple cycles currently in the (incomplete) cycle matrix $A = (a_{se})$; the entry a_{se} is 1 if the edge e participates in the simple cycle s, and 0 otherwise; ℓ_s is the length (the number of edges) of the simple cycle s. The last inequality excludes maximum cardinality matchings on all subgraphs induced by simple cycles, see Section 3.3.

The integer program (8) with the complete cycle matrix A can be fed to a general-purpose integer programming solver such as Gurobi [53] or SCIP [2] if enumerating all simple cycles of the bipartite graph B happens to be tractable; see [66] for enumerating all simple cycles in the directed case, and [77, Sec. 3.3] for the undirected case. These state-of-the-art integer programming solvers usually do not have any difficulty solving (8) to optimality. Unfortunately, enumerating all simple cycles is typically intractable in practice.

5. Notes on closely related problems.

5.1. Tearing in chemical engineering. As already stated in Section 3.4, the minimum feedback edge set problem and tearing are related but not equivalent problems. In particular, the input of tearing is an undirected bipartite graph, whereas the input of the minimum feedback edge set problem is a directed graph that is not necessarily bipartite.

Unfortunately, the term tearing is used in three different ways in the chemical engineering literature: It is sometimes used (1) exclusively for the (weighted) minimum feedback edge set problem, e.g., [14, 31, 46, 51, 78, 91, 97, 101, 121, 123, 133], and [16, Ch. 8], (2) for both the minimum feedback edge set problem and for tearing as in our terminology as defined above, see e.g. [59, 83, 89, 103], and (3) primarily in our sense, e.g., [19, 26, 52, 58, 79, 108, 110, 111, 129]. This issue seems to be specific to the

chemical engineering literature: For example, in the electrical engineering literature, tearing is always used in our sense.

The reason why the (weighted) minimum feedback edge set problem has received considerable attention in the field of chemical engineering is that it provides means to find favorable computation sequences in chemical process flowsheet calculations, see e.g. [5]. In this setup, the edges of the corresponding graph are already directed and fixed, and not individual equations are solved but small subsystems of equations (blocks, corresponding to the devices). However, if the direction of the edges is not considered fixed but can be chosen by the algorithm, the computational sequences obtained by solving the (weighted) minimum feedback edge set problem can become suboptimal compared to those obtained with tearing. This was known early on [25, 58, 78].

5.2. Further closely related problems. A **feedback vertex set** of a simple connected directed graph G is a set of vertices whose removal makes G acyclic; a feedback vertex set contains at least one vertex of every cycle in G. The term feedback vertex set also appears as **essential set** in the literature. Finding a feedback vertex set of minimum cardinality is the **minimum (directed) feedback vertex set problem**. The reductions between the minimum feedback edge set problem and the minimum feedback vertex set problem preserve feasible solutions and their cost; a good algorithm for one problem usually yields a good algorithm for the other [42].

The **linear ordering problem** can be defined as follows [85]. Given an $n \times n$ matrix A, determine a simultaneous permutation of its rows and columns such that the sum of the superdiagonal entries is maximal. This problem is also referred to as the **triangulation problem**.

6. Issues with the objective function of tearing. The true goal of tearing is to minimize the time that is necessary to solve the system of nonlinear equations (1). Unfortunately, this objective would be too complicated to handle accurately; compromises have to be made. A popular choice in the context of solving nonlinear systems of equations is to minimize the border width d, see Sec. 2.2. This objective function is used in the present paper too; here we discuss the issues associated with it. We ignore the effects of different software and hardware environments on the running time.

The most obvious issue is that minimizing d does not *guarantee* any savings in computation time. A convenient assumption is that minimizing d also minimizes the computation time, which may not be the case in practice.

Another issue is that the computational work saved with a particular ordering depends on the numerical method used to solve the final system (6), $H(z) = 0$, see for example [38, 92, 122, 128] and [16, Sec. 8.4]. Even for a given numerical method for solving $H(z) = 0$, and with d fixed, there can be several orderings that realize this fixed d, yet the computational savings can vary significantly.

A related problem is that $H(z) = 0$ can be very ill-conditioned, which in turn can have a negative impact on the convergence properties of the method used for solving it. This can become so severe that even when $f(x) = 0$ is well-conditioned, there may not be any machine representable value for z such that $H(z) = 0$ is satisfied with acceptable numerical accuracy, see for example the online supplement of [11]. Another form of ill-conditioning is that after recovering x from z, the original system $f(x) = 0$ is not satisfied even though $H(z) = 0$ holds within the pre-defined threshold. Although we focus on the nonlinear case, we mention that numerical issues are a major concern even in the linear case [3, 40, 41, 43].

All these issues are partly due to the fact that the nonlinearities and the actual

numerical values in (3) and (4) were intentionally ignored as a compromise to make the algorithm simpler. Although already the first papers on tearing mention that this compromise has the aforementioned drawbacks, see e.g. [26, 79, 115], the vast majority of the published tearing algorithms only consider the sparsity pattern of the Jacobian.

In the context of global optimization, [11, 13] offer means to mitigate all these issues through a so-called reparameterization. That method works well with extremely ill-conditioned torn systems even if both the nonlinearities and the actual numerical values were ignored when computing the permutation matrices in (2). The online supplement of [11] explains with an example how reparameterization mitigates the numerical issues of tearing.

7. Variants of tearing allowing small solvable subsystems. Our definition of tearing covers a common form of tearing, but other variants of tearing exist that may serve certain practical applications better. Sections 7.1 and 7.2 lay the groundwork for Sections 7.3 and 7.4: Desirable forms of sparse matrices are introduced in Sections 7.1 and 7.2 that differ from bordered lower triangular. The improvements discussed in Section 7.3 try to reduce the size of the final system (6) by relaxing the requirements of (3) (by allowing implicit equations for example) and/or allowing A in (4) to have a form other than lower triangular. All the enhancements discussed in Sections 7.3 and 7.4 share that the computation of y for a given z only involves fast and numerically stable algorithms such as solving implicit univariate equations or small systems of linear equations.

7.1. Dulmage-Mendelsohn decomposition and block triangular forms. The Dulmage-Mendelsohn decomposition was introduced in a collection of papers [35–37, 65]; more recent overviews of this decomposition method are available e.g., in [100], [34, Ch. 6], and [29, Ch. 7]. A computer implementation for performing the Dulmage-Mendelsohn decomposition is `HSL_MC79` from the Harwell Subroutine Library [61]. Only the relevant aspects of the Dulmage-Mendelsohn decomposition are summarized below; the reader is referred to the above references for further details.

The Dulmage-Mendelsohn decomposition starts with finding a maximum matching (see Fig. 4b). As discussed in Section 3.1, maximum matching corresponds to a permutation that places a maximum number of nonzero elements on the diagonal of the matrix. A (possibly rectangular) matrix has **structural full rank** if and only if the length of the maximum matching equals the length of the main diagonal. The matrix is **structurally singular** otherwise. One can efficiently find the structural rank of a given matrix with, for example, the `MC21` from the Harwell Subroutine Library [61]; this algorithm is described in [32, 33]. Although the Hopcroft-Karp algorithm [60] has better asymptotic worst-case complexity, numerical evidence suggests that `MC21` tends to outperform it in practice.

Given the input matrix A, the **coarse Dulmage-Mendelsohn decomposition** yields a row permutation P and a column permutation Q such that

$$PAQ = \begin{bmatrix} A_{11} & & & \\ A_{21} & & & \\ A_{31} & A_{32} & & \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix}, \tag{9}$$

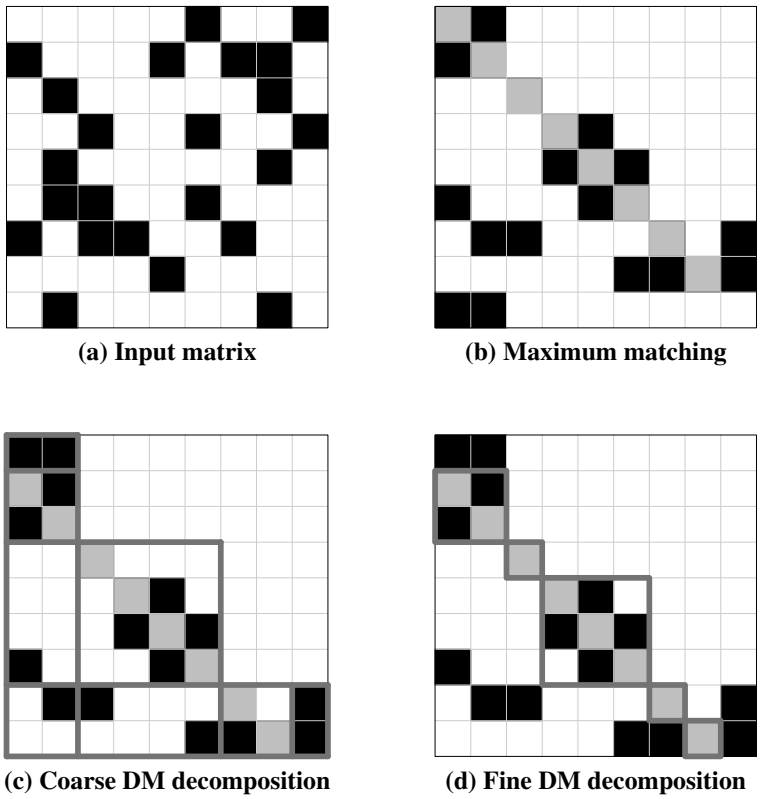


FIG. 4. *An example illustrating how the Dulmage-Mendelsohn decomposition (DM) works. The zero-free diagonals of the blocks are marked in gray.*

where

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \tag{10}$$

is either absent (has no rows and no columns) or it is rectangular and has more rows than columns; similarly,

$$\begin{bmatrix} A_{43} & A_{44} \end{bmatrix} \tag{11}$$

is either absent or it is rectangular and has more columns than rows; the blocks A_{21}, A_{32}, and A_{43} are square with a zero free-diagonal. The matrix (10) corresponds to the structurally overdetermined, and (11) to the structurally underdetermined part of the system $Ax = b$; both matrices are absent if the system $Ax = b$ is structurally well-defined. An example is shown in Figure 4c.

The coarse Dulmage-Mendelsohn decomposition is unique up to a certain degree. One can swap rows between A_{11} and A_{21}, or columns between A_{43} and A_{44}, as long as the matrices A_{21} and A_{43} still have a zero-free diagonal after the swaps, but apart from this, the row and column partition is unique. The order of the rows and the columns within each group of the partition is typically not unique: One can freely rearrange the rows and the columns within any of the blocks as long as the zero-free diagonals of the blocks A_{21}, A_{32}, and A_{43} are maintained.

The **fine Dulmage-Mendelsohn decomposition** yields a row permutation P and a column permutation Q such that A_{21}, A_{32}, and A_{43} have (possibly smaller) irreducible square blocks on the diagonal, with each block having a zero-free diagonal, furthermore A_{32} becomes block lower triangular as well. **Irreducible** in this context means that repeating the (coarse and the fine) Dulmage-Mendelsohn decomposition on these blocks would not decompose them further into smaller blocks.

The fine Dulmage-Mendelsohn decomposition is also unique up to a certain degree. It is sometimes the case that the irreducible blocks on the diagonal can be interchanged, and similarly to the coarse Dulmage-Mendelsohn decomposition, one can rearrange the rows and the columns within a block as long as the zero-free diagonal of the corresponding blocks is maintained.

The primary application area of tearing is the steady-state simulation of technical systems. A well-defined steady-state model has a square Jacobian with structural full rank; by default, any mainstream modeling environment will give an error and reject the model if this requirement is not satisfied. A special case of the Dulmage-Mendelsohn decomposition is the so-called **block lower triangular decomposition** or **BLT decomposition**: It outputs a block lower triangular form if the input matrix is square and structurally nonsingular (all submatrices of (9) are absent except A_{32}). Let $A \in \mathbb{R}^{n \times n}$ denote a square matrix that has structural full rank. The coarse Dulmage-Mendelsohn decomposition leaves A intact (i.e. only the trivial decomposition exists), and the fine Dulmage-Mendelsohn decomposition permutes A into block triangular form where each diagonal block is square with a zero-free diagonal and is irreducible.

In practice, a common approach to tearing is to perform a BLT decomposition first; it is also referred to as **partitioning and precedence ordering** in the chemical engineering literature. Tearing is then applied to the irreducible blocks on the diagonal. Some tearing heuristics even require BLT decomposition, such as Cellier's tearing [24, 116], otherwise the heuristic can fail. However, performing BLT decomposition first and applying tearing to the individual blocks can lead to suboptimal results [26, 110]. An example is shown in Figure 5. The matrix on the left of Figure 5

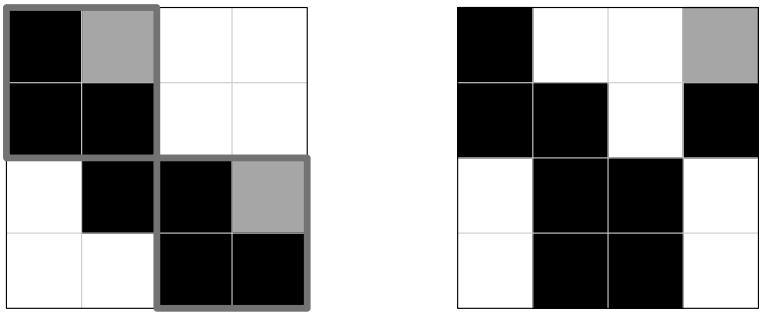


FIG. 5. *Left: Performing block lower triangular decomposition first, and then applying tearing to the two* 2×2 *blocks on the diagonal leads to suboptimal results. The variables that have to be guessed in each block are marked in gray. Right: The optimal ordering. The only guessed variable is marked in gray.*

is already in block lower triangular form with two 2×2 blocks on the diagonal. Applying tearing to these blocks consecutively results in 1 guessed variable in each block (marked in gray); the total cost is 2. However, if block lower triangular decomposition is not performed, and tearing is applied to the original matrix, the optimal cost

tearing is 1 (the only guessed variable is marked in gray).

7.2. Desirable forms of sparse matrices. In this section, we discuss the desirable forms of square and irreducible sparse matrices in terms of the sparsity pattern. The matrices are assumed to be highly unsymmetric. Highly unsymmetric in this context means that introducing artificial nonzero entries to make the sparsity pattern symmetric is not acceptable from the point of view of the application. Such matrices often arise in engineering applications, and these are probably the primary area of application for the proposed methods of the present paper.

In a **bordered block lower triangular** form, that is, in

$$A = \begin{bmatrix} L & B \\ C & D \end{bmatrix} \tag{12}$$

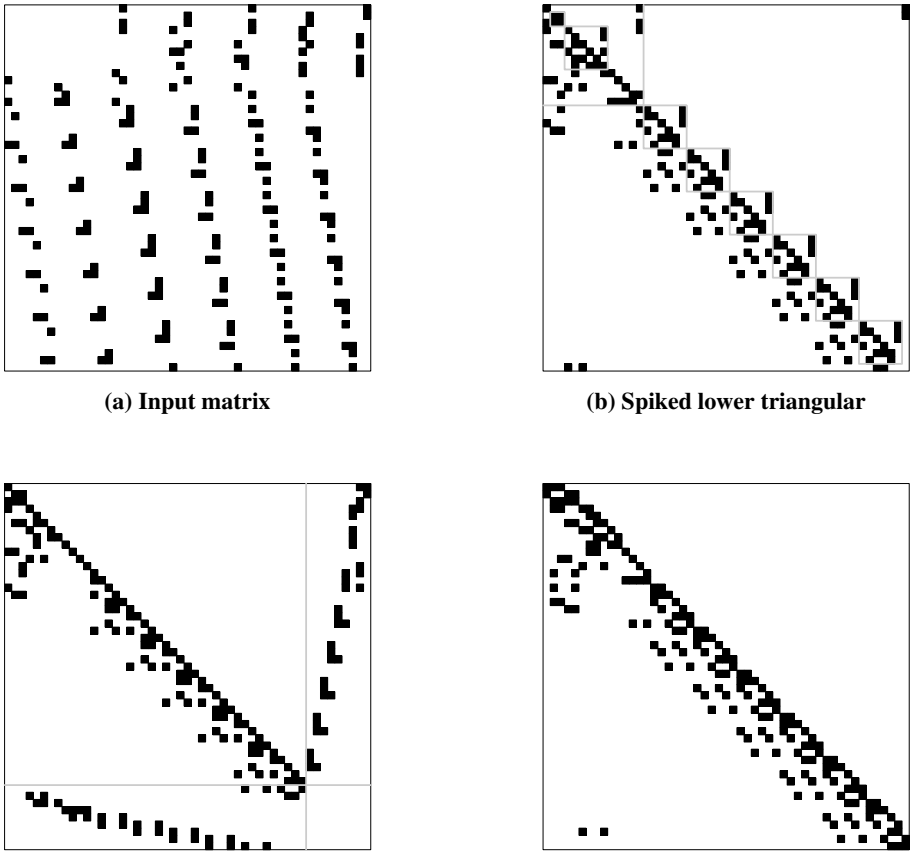
the leading submatrix L is a block lower triangular matrix whose diagonal blocks are square and structurally nonsingular. An example is shown in Figure 6, where the input matrix (Fig. 6a) corresponds to the steady-state model equations of the distillation column of [64] (with $N = 8$ stages). There is a 2×2 block on the diagonal of L; the submatrix D happens to be empty (Fig. 6c).

A **spiked lower triangular** matrix is a nearly lower triangular matrix where some of the columns have entries above the diagonal; these columns are called **spike** columns or simply spikes. The diagonal entry in a non-spike column must be nonzero, however, a spike column can have a zero entry on the diagonal. Furthermore, for any pair of spike columns, referred to as left and right spike below, the following property must hold for the entries on and above the diagonal: The set of rows in the left spike is either contained in or disjoint from the set of rows of the right spike (see Fig. 6b). This requirement means that we can form bordered block lower triangular forms on the diagonal at the spikes where the blocks are either properly nested or disjoint, see Figure 6b. These diagonal blocks themselves are bordered block lower triangular forms recursively; therefore the spiked form is also referred to as **nested bordered block triangular form**, see [34, Sec. 8.9].

A **lower Hessenberg form** is a block lower triangular matrix but with fully dense rectangular blocks on the diagonal, rather than square blocks. Consequently, the height of the columns is nonincreasing. Since the matrix is irreducible, the first entry in each column is either on or above the diagonal. An example is shown in Figure 6d. A lower Hessenberg form can be transformed to a spiked form, and also to a bordered block lower triangular form with a simple single-pass algorithm [43].

Practical algorithms for permuting to these forms. Several heuristics have been proposed to permute A to one of the desirable forms discussed in this section, e.g., the Hellerman-Rarick family of ordering algorithms, see [40, 55, 56] and [34, Ch. 8], and the ordering algorithms of Stadtherr and Wood [108, 109]. Efficient computer implementation of the Hellerman-Rarick family of ordering algorithms is `MC33` from the Harwell Subroutine Library [61]. Although there are subtle differences among these ordering algorithms, they all fit the same pattern when viewed from a sufficiently high level of abstraction [43]; they only seem to differ in the lookahead heuristics applied to break the ties when picking the next row to be eliminated. This abstraction by Fletcher and Hall [43] is discussed in [9] in great detail.

7.3. Hierarchical tearing. In certain cases, it can be beneficial to allow J to be a spiked form, and minimize the number of spike columns. When dealing with spiked forms, the number of spike columns plays the role of d since that determines the size of the final system (6). (Another way to look at this is to consider the bordered lower



(a) Input matrix **(b) Spiked lower triangular**

(c) Bordered block lower triangular **(d) Lower Hessenberg**

FIG. 6. *Alternative forms for ordered sparse matrices.*

triangular form as the special case of the spiked form where all the spikes are situated at the border.)

Tearing can be applied in a recursive fashion if J is allowed to be spiked lower triangular with disjoint (i.e. unnested) blocks. In this case, each block on the diagonal is bordered lower triangular by definition of the spiked form; therefore tearing can be applied successively to the blocks along the diagonal.

An example is shown in Figure 7. The original system is on the left of the figure; it is an 8×8 system and has two blocks on the diagonal. Since both of these blocks are bordered lower triangular, we can apply tearing successively to them to get a sequence of two univariate equations; this is shown on the right of Figure 7. For the sake of this example we assume that these univariate equations are implicit nonlinear equations, and can be solved numerically for the variables corresponding originally to the spike columns. (Note that this is a significant deviation from the definition of Section 2.1 where we disallowed implicit equations in (3).) Since the resulting 4×4 system is bordered lower triangular, we can apply tearing again to get a 2×2 final system. We can pose this final system to a nonlinear system solver as follows: (i) The two variables on the border are posed to the solver (see on the right of Figure 7); (ii) we deal with the sequence of the two implicit equations internally; (iii) return the

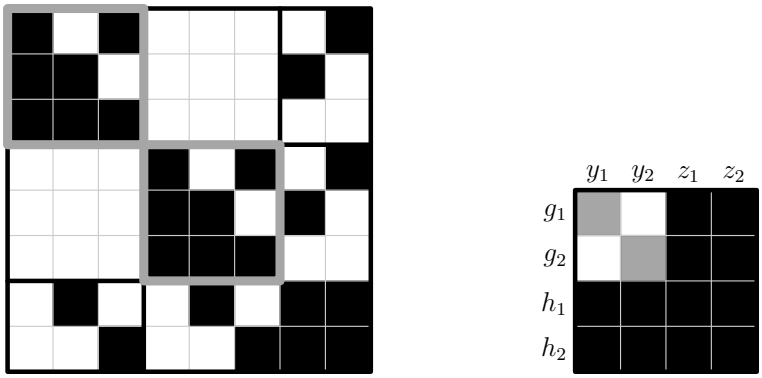


FIG. 7. *Left: The original* 8×8 *system; tearing will be applied to the marked two* 3×3 *blocks on the diagonal to get a sequence of two implicit equations. Right: The gray* 1×1 *blocks are the result. This reduced* 4×4 *system can be posed to a solver as the* 2×2 *system* $h(z) = 0$ *by applying tearing again, and solving the implicit* g_i *for* y_i $(i = 1, 2)$ *internally.*

residual of the last two equations as the residual of the system.

This approach is known as the stage-by-stage method [80, 117] in the chemical engineering literature. Distillation columns can greatly benefit from this improvement of tearing: the size of the final system corresponding to (6) becomes independent of the problem size, whereas it would grow linearly with the problem size if spikes were not allowed in J. To the best of our knowledge, the above description is the first abstract presentation of the stage-by-stage method in terms of the sparsity pattern.

The requirement mentioned earlier, that the blocks must be disjoint, is not necessary, although the presence of nesting makes the recursive application of tearing less appealing. If the nested blocks are solved from inside out and sequentially, then the method will resemble the coordinate search method (see [93, Sec. 9.3]) in the coordinate system of the variables corresponding to the spikes, and with a fixed order of the search directions. This can be quite inefficient in practice. If the nested blocks are merged by moving the spikes of the inner blocks to the border of the outermost block to form a single bordered lower triangular block (see Sec. 7.2), then this final block will potentially have a thick border; nevertheless, the corresponding reduced system of equations can still be solved with any state-of-the-art trust-region or line search method [93, Sec. 11.2]. To summarize, a spiked lower triangular form for J is appealing if the blocks are disjoint and have a very narrow border (preferably width 1). We will later give further examples of hierarchical tearing in other contexts as well, in Section 10.1 (fill-reducing orderings).

7.4. Identifying linear or other small solvable subsystems. Depending on the problem structure, it can be advantageous to allow A in (4) to be block lower triangular but require that the blocks on the diagonal correspond to linear systems of equations [39]. Such diagonal blocks can be solved efficiently.

Technical systems must obey the conservation laws (mass and heat balances must hold), and the corresponding equations are intrinsically linear. Sophisticated ordering rules for forming linear subsystems and small subblocks were proposed in [58, 108, 110]. We note that, when viewed from the solvable subsystems point of view, the hierarchical tearing of Section 7.3 forms solvable subsystems (namely the blocks) recursively.

8. Improvements for heuristic-based tearing methods.

8.1. Improved matching. Since an exact algorithm must consider all possible orientations, improved matchings are interesting mostly in the context of greedy tearing heuristics. The problems listed in Section 6 (that stem from considering the sparsity pattern only) can be mitigated by taking into account the ignored information to some extent. For example in [128], not only the sparsity pattern of the Jacobian is considered but also the actual numerical values. The algorithm requires an estimate for the solution vector of (1). Another method is described in [52] that works as follows. According to some heuristic, a weight is assigned to each variable-equation pair, reflecting how desirable it is to solve that equation for that variable (match that variable with that equation). The heuristic can take into account the actual numerical values and the symbolic form of the equations. Then, either the maximum weight matching is computed, or the matching that maximizes the minimum desirability associated with any of the equation-variable matches. (See also `MC64` from the the Harwell Subroutine Library [61].) This matching is used for orienting the bipartite graph B corresponding to the Jacobian of the system, see Section (3.2). The algorithm may produce suboptimal orderings since the cycles in B are completely ignored when the matching is computed.

8.2. Exploiting the natural block structure. Most technical systems have an inherent block structure, corresponding to the devices of the system. This block structure can be used, for example, to partition the irreducible blocks further, to pre-order the rows and the columns of the Jacobian, or to get a coarse decomposition. Numerical evidence suggests that the partition obtained from the inherent block structure is useful for heuristic orderings in several contexts: (1) fill-reducing orderings [73, 108, 109, 130], c.f. Section 10.1, (2) parallelization of the solution process of sparse linear systems and DAE systems [1, 20, 113, 124, 126], c.f. Section 10.2, (3) ordering for global optimization [11–13]. The necessary information on the block structure is available inside object-oriented simulators such as Modelica [45, 86, 118] for multi-domain modeling of heterogeneous complex technical systems, and gPROMS [47], ASCEND [98] and EMSO [30] for chemical process modeling, simulation and optimization.

9. Improved exact algorithms.

9.1. Lazy constraint generation. The reason why solving the integer program (8) can become impractical is that the naive implementation requires enumerating all the simple cycles in B. Unfortunately, even sparse graphs can have exponentially many simple cycles [104], and such graphs appear in practice, e.g., cascades (distillation columns) can realize this many simple cycles. In a more sophisticated setting, one may try to guess the necessary constraints in (8), and enumerate only the corresponding simple cycles. The hope is that only polynomially many simple cycles will be necessary. In the context of tearing, the idea of lazy constraint generation appears in [89, 90].

The idea of building up an integer program incrementally, by adding constraints to it in a lazy fashion, is quite old, see for example Dantzig et al. [28] from 1954. The well-known column generation approach corresponds to this idea but works on the dual problem. Probably the first published paper applying column generation is from 1958 by Ford and Fulkerson [44]. Not surprisingly, state-of-the-art integer programming solvers have high-level API support for implementing such algorithms, see for example `LazyConstraints` in Gurobi 6.5.1 [53].

9.2. Smart enumeration. Smart enumeration algorithms resemble branch and bound methods, but they do not consider themselves as such. With this type of exact methods all possibilities are enumerated, however, in a smart way. After having enumerated and evaluated the goodness of an ordering, the so-called exclusion rules help to exclude certain orderings from the yet to be enumerated orderings, significantly reducing the combinations to be examined. Care is taken such that the better orderings are likely to be enumerated earlier. The basis of the exclusion rule is that it is not necessary to enumerate and evaluate an ordering if we can prove that that ordering cannot be strictly better then the currently best known one. Sophisticated examples of these algorithms are given in [26] and [58].

10. Context dependent alternative objective functions. Several issues were pointed out in Section 6 concerning the objective function. In this section we present three other objective functions that are used in other contexts with the intent of minimizing the running time. However, neither of these other objective functions actually guarantee minimal running time either.

10.1. Fill-reducing orderings. In the context of direct methods for solving sparse systems of linear equations (e.g. sparse LU factorization), a common objective function for tearing is to reduce the fill-in. Under reasonable assumptions, minimizing this objective is also expected to minimize the running time of the algorithm for solving the corresponding linear system, c.f. [50]. However this objective also has issues, just like minimizing the border width had; the truly "best" ordering cannot be defined precisely [34, p. 127]. For this reason, and because minimizing the fill-in is NP-complete [102, 132], heuristics are used in practice: Hierarchical tearing (e.g. [81] but see also Section 7.3), and exploiting the inherent block structure proved to be successful, c.f. Section 8.2. Some examples of fill-reducing algorithms from the fields of chemical and electrical engineering are: [73, 108, 109, 111, 112, 119, 130]. This list is certainly not exhaustive.

An interesting approach in this area is the combination of nested dissection and minimum degree ordering algorithms [48, 57, 96]. Numerical evidence suggests that nested dissection performs better on larger problems, while minimum degree ordering produces less fill-in for smaller ones. Therefore, a hybrid heuristic is proposed, applying nested dissection first, followed by minimum degree ordering within the obtained smallest blocks.

Tearing is not a fill-reducing ordering: When breaking ties, tearing can make the exact opposite decision of what a fill-reducing ordering (e.g. Markowitz rule [84]) would make [43].

10.2. Parallel computations. In the context of parallel computations, the goal is to decompose the system in such a way that the subproblems can be solved in any order (in parallel). The corresponding desirable sparsity pattern is the recursive bordered block diagonal form (RBBD).

The idea of parallel decompositions dates back to the work of Kron [70, 71]. Elaborate decomposition methods have been published and successfully applied to (chemical and electrical) engineering problems, see for example [1, 4, 20, 27, 62, 113, 124, 126, 127]. The ordering heuristics often exploit the inherent block structure of the technical system, c.f. Section 8.2, or apply the hierarchical tearing approach, see Section 7.3. A formula is proposed in [4] to decide whether applying tearing recursively to a subblock is worthwhile or not.

10.3. Minimizing the largest subsystem size. Minimizing the largest subsystem size in the decomposed system appeared already in the first publications on tearing [115]. This is a computationally challenging task: Even if the size of such a largest subsystem is bounded by a constant, the problem is still not solvable in polynomial time (under the hypothesis that $W[1] \neq FPT$), see [18]. Nevertheless, minimizing the size of the largest *nonlinear* subsystem seems to be very appealing in the context of global optimization [11–13]; partitioning based on the natural block structure, c.f. Section 8.2, can help to create an efficient greedy heuristic.

11. Other applications.

Initializing and solving DAE systems. Numerous (primarily chemical and electrical) engineering applications have been referenced in the previous sections in various contexts. The reason is the following: The problem of solving nonlinear systems of equations arises in the daily engineering practice, e.g., when consistent initial values for differential algebraic equation (DAE) systems are sought [95, 120], or when solving steady-state models of technical systems. A steady-state solution can be used as a consistent initial set of the DAE system [72].

Often, steady-state initialization failures can only be resolved in very cumbersome ways [8, 94, 105, 106, 125], involving user-provided good initial values for the variables. However, [94] reports that tearing proved to be helpful in steady-state initialization in certain, difficult cases. Tearing usually also helps to speed up the solution process of DAE systems [23, 24, 38, 39, 87], thanks to either the reduced problem size or parallelization, or both. Computing multiple stead-states with tearing-based decomposition was proposed in [11–13].

Design structure matrix. The so-called design structure matrix (DSM) provides a concise visual representation of a complex systems. DSMs can help to decompose, manage, and integrate design processes of complex systems [21, 74, 75, 114], or to explore the structure of complex software designs [82]. Here, partitioning and tearing provides means to decompose a complex problem into subproblems.

The industrial applications in the 1990's include aerospace [49, 107], automotive [17], building design [6, 7], manufacturing [76], telecommunications [99] applications. The recent survey of Browning [22], entitled "Design Structure Matrix Extensions and Innovations: A Survey and New Opportunities" cites 553 papers, covering the following areas: aerospace, automotive, computer (hardware), construction, electronics, energy, government agencies, health-care, information systems and technologies, innovation systems, manufacturing systems, mechanical products/equipment, military, naval ship design and development, network system control, pharmaceutical, real estate development, sensor systems (large-scale), service system design, software development, transportation system organizations.

Economics. Thanks to the typical structure of a macroeconomic model, tearing can significantly reduce the size of the problem which in turn can result in significant savings in the computational efforts necessary to solve these models [15, 63].

Acknowledgement. The research was funded by the Austrian Science Fund (FWF): P27891-N32. In addition, support by the Austrian Research Promotion Agency (FFG) under project number 846920 is gratefully acknowledged.

References.

[1] K. A. Abbott, B. A. Allan, and A. W. Westerberg. Global preordering for Newton equations using model hierarchy. *AIChE J.*, 43:3193–3204, 1997.

[2] Tobias Achterberg. SCIP: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009.

[3] M. Arioli, I. S. Duff, N. I. M. Gould, and J. K. Reid. Use of the P^4 and P^5 algorithms for in-core factorization of sparse matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):913–927, 1990.

[4] Hideki Asai and Shinsaku Mori. Network analysis by hierarchical tearing method. *Systems and Computers in Japan*, 16(6):59–67, 1985.

[5] Aspen Technology, Inc. Aspen Simulation Workbook, Version Number: V7.1, 2009. Burlington, MA, USA. EO and SM Variables and Synchronization, p. 110.

[6] Simon Austin, Andrew Baldwin, and Andrew Newton. Manipulating the flow of design information to improve the programming of building design. *Construction Management and Economics*, 12(5):445–455, 1994.

[7] Simon Austin, Andrew Baldwin, Baizhan Li, and Paul Waskett. Analytical design planning technique (ADePT): a dependency structure matrix tool to schedule the building design process. *Construction Management and Economics*, 18(2):173–182, 2000.

[8] B. Bachmann, P. Aronßon, and P. Fritzson. Robust initialization of differential algebraic equations. In *1st International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (Berlin; Germany; July 30; 2007)*, Linköping Electronic Conference Proceedings, pages 151–163. Linköping University Electronic Press; Linköpings universitet, 2007.

[9] A. Baharev, H. Schichl, and A. Neumaier. Ordering matrices to bordered lower triangular form with minimal border width. Submitted, 2016.

[10] A. Baharev, H. Schichl, A. Neumaier, and T. Achterberg. An exact method for the minimum feedback arc set problem. Submitted, 2016.

[11] Ali Baharev and Arnold Neumaier. A globally convergent method for finding all steady-state solutions of distillation columns. *AIChE J.*, 60:410–414, 2014.

[12] Ali Baharev, Lubomir Kolev, and Endre Rév. Computing multiple steady states in homogeneous azeotropic and ideal two-product distillation. *AIChE Journal*, 57:1485–1495, 2011.

[13] Ali Baharev, Ferenc Domes, and Arnold Neumaier. Sampling solutions of sparse nonlinear systems. Submitted, 2016. URL http://www.mat.univie.ac.at/~neum/ms/maniSol.pdf.

[14] R. W. Barkley and R. L. Motard. Decomposition of nets. *Chem. Eng. J.*, 3: 265–275, 1972.

[15] Robin Becker and Berc Rustem. Algorithms for solving nonlinear dynamic decision models. *Annals of Operations Research*, 44(2):115–142, 1993.

[16] Lorenz T. Biegler, Ignacio E. Grossmann, and Arthur W. Westerberg. *Systematic Methods of Chemical Process Design*. Prentice Hall PTR, Upper Saddle River, NJ, 1997.

[17] Thomas A Black, Charles H Fine, and Emanuel M Sachs. A method for systems design using precedence relationships: An application to automotive brake systems. Working Paper 3208, Sloan School of Management, MIT, Cambridge, MA, 1990.

[18] Matthijs Jacobus Bomhoff. *Bipartite graphs and the decomposition of systems of equations*. PhD thesis, Enschede, January 2013. URL http://doc.utwente.nl/83192/.

[19] N. L. Book and W. F. Ramirez. Structural analysis and solution of systems of algebraic design equations. *AIChE Journal*, 30(4):609–622, 1984.

[20] Jürgen Borchardt. Newton-type decomposition methods in large-scale dynamic process simulation. *Computers and Chemical Engineering*, 25:951–961, 2001.

[21] T. R. Browning. Applying the design structure matrix to system decomposition and integration problems: a review and new directions. *Engineering Management, IEEE Transactions on*, 48(3):292–306, 2001.

[22] T.R. Browning. Design structure matrix extensions and innovations: A survey and new opportunities. *Engineering Management, IEEE Transactions on*, 63 (1):27–52, 2016.

[23] Emanuele Carpanzano. Order reduction of general nonlinear DAE systems by automatic tearing. *Mathematical and Computer Modelling of Dynamical Systems*, 6(2):145–168, 2000.

[24] François E Cellier and Ernesto Kofman. *Continuous system simulation*. Springer Science & Business Media, 2006.

[25] J. H. Christensen and D. F. Rudd. Structuring design computations. *AIChE Journal*, 15:94–100, 1969.

[26] James H. Christensen. The structuring of process optimization. *AIChE Journal*, 16(2):177–184, 1970.

[27] A. B. Coon and M. A. Stadtherr. Generalized block-tridiagonal matrix orderings for parallel computation in process flowsheeting. *Computers & Chemical Engineering*, 19(67):787 – 805, 1995.

[28] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2 (4):393–410, 1954.

[29] Timothy A. Davis. Direct methods for sparse linear systems. In Nicholas J. Higham, editor, *Fundamentals of algorithms*. Philadelphia, USA: SIAM, 2006.

[30] R. de P. Soares and A. R. Secchi. EMSO: A new environment for modelling, simulation and optimisation. In *Computer Aided Chemical Engineering*, volume 14, pages 947–952. Elsevier, 2003.

[31] A. C. Dimian, C. S. Bildea, and A. A. Kiss. Chapter 3: Steady-state flowsheeting. In *Integrated Design and Simulation of Chemical Processes*, volume 35 of *Computer-Aided Chemical Engineering*. Elsevier Amsterdam, 2nd edition, 2014.

[32] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Software*, 7:315–330, 1981.

[33] I. S. Duff. Algorithm 575: Permutations for a zero-free diagonal. *ACM Trans. Math. Software*, 7:387–390, 1981.

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

[35] A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Can. J. Math.*, 10:517–534, 1958.

[36] A. L. Dulmage and N. S. Mendelsohn. A structure theory of bipartite graphs of finite exterior dimension. *Trans. Royal Society of Canada. Sec. 3.*, 53:1–13, 1959.

[37] A. L. Dulmage and N. S. Mendelsohn. Two Algorithms for Bipartite Graphs. *J. Soc. Ind. Appl. Math.*, 11:183–194, 1963.

[38] H. Elmqvist and M. Otter. Methods for tearing systems of equations in object-oriented modeling. In *Proceedings ESM'94, European Simulation Multiconference, Barcelona, Spain, June 1–3*, pages 326–332, 1994.

[39] Hilding Elmqvist. *A Structured Model Language for Large Continuous Systems*. PhD thesis, Department of Automatic Control, Lund University, Sweden, May 1978.

[40] A. M. Erisman, R. G. Grimes, J. G. Lewis, and W. G. Jr. Poole. A structurally stable modification of Hellerman-Rarick's P^4 algorithm for reordering unsymmetric sparse matrices. *SIAM J. Numer. Anal.*, 22:369–385, 1985.

[41] A. M. Erisman, R. G. Grimes, J. G. Lewis, Jr. W. G. Poole, and H. D. Simon. Evaluation of orderings for unsymmetric sparse matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(4):600–624, 1987.

[42] G. Even, J. (Seffi) Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20(2):151–174, 1998.

[43] R. Fletcher and J. A. J. Hall. Ordering algorithms for irreducible sparse linear systems. *Annals of Operations Research*, 43:15–32, 1993.

[44] L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[45] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1*. Wiley-IEEE Press, 2004.

[46] P. L. Genna and R. L. Motard. Optimal decomposition of process networks. *AIChE Journal*, 21(4):656–663, 1975.

[47] gPROMS. Process Systems Enterprise Limited, gPROMS. http://www.psenterprise.com, 2015. [Online; accessed 29-Feb-2016].

[48] Laura Grigori, Erik G. Boman, Simplice Donfack, and Timothy A. Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse LU factorization. *SIAM Journal on Scientific Computing*, 32(6):3426–3446, 2010.

[49] D. Grose. Reengineering the aircraft design process. In *5th Symposium on Multidisciplinary Analysis and Optimization*, 1994. URL http://dx.doi.org/10.2514/6.1994-4323.

[50] G. Guardabassi and A. Sangiovanni-Vincentelli. A two levels algorithm for tearing. *Circuits and Systems, IEEE Transactions on*, 23(12):783–791, 1976.

[51] T. Gundersen and T. Hertzberg. Partitioning and Tearing of Networks Applied to Process Flowsheeting. *Modeling, Identification and Control*, 4(3):139–165, 1983.

[52] Prem K. Gupta, Arthur W. Westerberg, John E. Hendry, and Richard R. Hughes. Assigning output variables to equations using linear programming. *AIChE Journal*, 20(2):397–399, 1974.

[53] Gurobi. Gurobi Optimizer Version 6.0. Houston, Texas: Gurobi Optimization, Inc., May 2015. (software program). http://www.gurobi.com, 2014.

[54] V. Guruswami, R. Manokaran, and P. Raghavendra. Beating the random ordering is hard: Inapproximability of maximum acyclic subgraph. In *Foundations of Computer Science, 2008. FOCS '08. IEEE 49th Annual IEEE Symposium on*, pages 573–582, 2008.

[55] E. Hellerman and D. C. Rarick. Reinversion with preassigned pivot procedure. *Math. Programming*, 1:195–216, 1971.

[56] E. Hellerman and D. C. Rarick. The partitioned preassigned pivot procedure (P^4). In Donald J. Rose and Ralph A. Willoughby, editors, *Sparse Matrices and their Applications*, The IBM Research Symposia Series, pages 67–76. Springer US, 1972.

[57] Bruce Hendrickson and Edward Rothberg. Improving the run time and quality of nested dissection ordering. *SIAM Journal on Scientific Computing*, 20(2): 468–489, 1998.

[58] R. Hernandez and R. W. H. Sargent. A new algorithm for process flowsheeting. *Computers & Chemical Engineering*, 3(14):363–371, 1979.

[59] V. Hlaváček. Analysis of a complex plant-steady state and transient behavior. *Computers & Chemical Engineering*, 1(1):75 – 100, 1977.

[60] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[61] HSL. A collection of Fortran codes for large scale scientific computation., 2013. URL http://www.hsl.rl.ac.uk.

[62] Y. F. Hu, K. C. F. Maguire, and R. J. Blake. A multilevel unsymmetric matrix ordering algorithm for parallel process simulation. *Computers & Chemical Engineering*, 23(1112):1631 – 1647, 2000.

[63] Andrew Hughes Hallett and Paul Fisher. On economic structures and model solution methods: Or should econometricians use newton methods for model solution? *Oxford Bulletin of Economics and Statistics*, 52(3):317–30, 1990.

[64] E.W. Jacobsen and S. Skogestad. Multiple steady states in ideal two-product distillation. *AIChE Journal*, 37:499–511, 1991.

[65] D. M. Johnson, A. L. Dulmage, and N. S. Mendelsohn. Connectivity and reducibility of graphs. *Can. J. Math*, 14:529–539, 1962.

[66] Donald B. Johnson. Finding all the elementary circuits of a directed graph. *SIAM Journal on Computing*, 4(1):77–84, 1975.

[67] Viggo Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology Stockholm, 1992.

[68] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[69] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 767–775, New York, NY, USA, 2002. ACM.

[70] Gabriel Kron. A set of principles to interconnect the solutions of physical systems. *Journal of Applied Physics*, 24(8):965–980, 1953.

[71] Gabriel Kron. *Diakoptics: the piecewise solution of large-scale systems.* MacDonald, London, 1963.

[72] A. Kröner, W. Marquardt, and E.D. Gilles. Getting around consistent initialization of DAE systems? *Computers & Chemical Engineering*, 21(2):145–158, 1997.

[73] M. Kubíček, V. Hlaváček, and F. Procháska. Global modular Newton-Raphson technique for simulation of an interconnected plant applied to complex rectification columns. *Chemical Engineering Science*, 31(4):277 – 284, 1976.

[74] A. Kusiak and J. Wang. Efficient organizing of design activities. *International Journal of Production Research*, 31(4):753–769, 1993.

[75] A. Kusiak and J. Wang. Decomposition of the design process. *Journal of Mechanical Design*, 115(4):687–695, 1993.

[76] Andrew Kusiak, T. Nick Larson, and Juite (Ray) Wang. Reengineering of design and manufacturing processes. *Computers & Industrial Engineering*, 26(3):521–536, 1994.

[77] B. C. Lee. Algorithmic approaches to circuit enumeration problems and applications. Technical report, MIT, Dept. of Aeronautics & Astronautics, Flight Transportation Laboratory; Cambridge, MA, USA, 1982.

[78] W. Lee and D. F. Rudd. On the ordering of recycle calculations. *AIChE Journal*, 12(6):1184–1190, 1966.

[79] W. Lee, J. H. Christensen, and D. F. Rudd. Design variable selection to simplify

process calculations. *AIChE Journal*, 12(6):1104–1115, 1966.
[80] W. K. Lewis and G. L. Matheson. Studies in distillation. *Ind. Eng. Chem.*, 24: 494–498, 1932.
[81] T. D. Lin and R. S. H. Mah. Hierarchical partition-a new optimal pivoting algorithm. *Mathematical Programming*, 12(1):260–278, 1977.
[82] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Management Science*, 52(7):1015–1030, 2006.
[83] Richard S. H. Mah. 4 - Computation Sequence in Process Flowsheet Calculations. In Richard S. H. Mah, editor, *Chemical Process Structures and Information Flows*, pages 125–183. Butterworth-Heinemann, 1990.
[84] H. M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3(3):255–269, 1957.
[85] Rafael Martí and Gerhard Reinelt. *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*, volume 175 of *Applied Mathematical Sciences*. Springer-Verlag Berlin Heidelberg, 2011.
[86] S. Mattsson, H. Elmqvist, and M. Otter. Physical system modeling with Modelica. *Control. Eng. Pract.*, 6:501–510, 1998.
[87] S. E. Mattsson, M. Otter, and H. Elmqvist. Modelica hybrid modeling and efficient simulation. In *Decision and Control, 1999. Proceedings of the 38th IEEE Conference on*, volume 4, pages 3502–3507, 1999.
[88] Modelon AB. JModelica.org User Guide, verison 1.15. http://www.jmodelica.org/page/236, 2015. [Online; accessed 17-April-2015].
[89] J. M. Montagna and O. A. Iribarren. Optimal computation sequence in the simulation of chemical plants. *Computers & Chemical Engineering*, 12(1):71–79, 1988.
[90] J. M. Montagna and O. A. Iribarren. Optimal resolution sequence of problems modeled by directed graphs. *Mathematical and Computer Modelling*, 10(7): 515–521, 1988.
[91] Rodolphe L. Motard and Arthur W. Westerberg. Exclusive tear sets for flowsheets. *AIChE Journal*, 27:725–732, 1981.
[92] Rodolphe L. Motard and Arthur W. Westerberg. Exclusive tear sets for flowsheets. *AIChE Journal*, 27:725–732, 1981.
[93] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, USA, second edition, 2006.
[94] L. A. Ochel and B. Bachmann. Initialization of equation-based hybrid models within OpenModelica. In *5th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (University of Nottingham; Nottingham, UK; April 19, 2013)*, Linköping Electronic Conference Proceedings, pages 97–103. Linköping University Electronic Press; Linköpings universitet, 2013.
[95] C. C. Pantelides. The consistent initialization of differential-algebraic systems. *SIAM Journal on Scientific and Statistical Computing*, 9(2):213–231, 1988.
[96] François Pellegrini, Jean Roman, and Patrick Amestoy. *Hybridizing nested dissection and halo approximate minimum degree for efficient sparce matrix ordering*, pages 986–995. Springer Berlin Heidelberg, 1999.
[97] T. K. Pho and L. Lapidus. Topics in computer-aided design: Part I. An optimum tearing algorithm for recycle systems. *AIChE Journal*, 19(6):1170–1181, 1973.
[98] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. ASCEND: An object-oriented computer environment for modeling and analysis: The mod-

eling language. *Computers & Chemical Engineering*, 15(1):53–72, 1991.

[99] Randal D. Pinkett. Product development process modeling and analysis of digital wireless telephones. Master's thesis, Massachusetts Institute of Technology. Dept. of Electrical Engineering and Computer Science; Sloan School of Management, MIT, Cambridge, MA, 1998. URL http://hdl.handle.net/1721.1/9863.

[100] Alex Pothen and Chin-Ju Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Softw.*, 16:303–324, 1990.

[101] J. R. Roach, B. K. O'Neill, and D. A. Hocking. A new synthetic method for stream tearing in process systems analysis. *Chemical Engineering Communications*, 161(1):1–14, 1997.

[102] Donald J. Rose and Robert Endre Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM Journal on Applied Mathematics*, 34(1):176–197, 1978.

[103] R. W. H. Sargent. The decomposition of systems of procedures and algebraic equations. In G. A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 158–178. Springer Berlin Heidelberg, 1978.

[104] Benno Schwikowski and Ewald Speckenmeyer. On enumerating all minimal solutions of feedback problems. *Discrete Applied Mathematics*, 117(13):253–265, 2002.

[105] M. Sielemann and G. Schmitz. A quantitative metric for robustness of nonlinear algebraic equation solvers. *Mathematics and Computers in Simulation*, 81(12): 2673–2687, 2011.

[106] M. Sielemann, F. Casella, and M. Otter. Robustness of declarative modeling languages: Improvements via probability-one homotopy. *Simulation Modelling Practice and Theory*, 38:38–57, 2013.

[107] J. Sobieszczanski-Sobieski and R. T. Haftka. Multidisciplinary aerospace design optimization: survey of recent developments. *Structural optimization*, 14(1): 1–23, 1997.

[108] M. A. Stadtherr and E. S. Wood. Sparse matrix methods for equation-based chemical process flowsheeting–I: Reordering phase. *Computers & Chemical Engineering*, 8(1):9–18, 1984.

[109] M. A. Stadtherr and E. S. Wood. Sparse matrix methods for equation-based chemical process flowsheeting–II: Numerical Phase. *Computers & Chemical Engineering*, 8(1):19–33, 1984.

[110] M. A. Stadtherr, W. A. Gifford, and L. E. Scriven. Efficient solution of sparse sets of design equations. *Chemical Engineering Science*, 29(4):1025–1034, 1974.

[111] Mark A. Stadtherr. Maintaining sparsity in process design calculations. *AIChE Journal*, 25(4):609–615, 1979.

[112] Mark A. Stadtherr and E. Stephen Wood. Exploiting border structure in solving large sparse linear systems in bordered triangular form. *Computers & Chemical Engineering*, 4(3):191 – 199, 1980.

[113] G. N. Stenbakken and J. A. Starzyk. Diakoptic and large change sensitivity analysis. *IEE Proceedings G (Circuits, Devices and Systems)*, 139:114–118(4), February 1992.

[114] D. V. Steward. The design structure system: A method for managing the design of complex systems. *Engineering Management, IEEE Transactions on*, EM-28 (3):71–74, 1981.

[115] Donald V. Steward. Partitioning and tearing systems of equations. *Journal of the Society for Industrial and Applied Mathematics Series B Numerical Analysis*, 2(2):345–365, 1965.

[116] P. Täuber, L. Ochel, W. Braun, and B. Bachmann. Practical realization and adaptation of Cellier's tearing method. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 11–19, New York, NY, USA, 2014. ACM.

[117] E.W. Thiele and R.L. Geddes. Computation of distillation apparatus for hydrocarbon mixtures. *Ind. Eng. Chem.*, 25:289–295, 1933.

[118] M. Tiller. *Introduction to physical modeling with Modelica*. Springer Science & Business Media, 2001.

[119] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55: 1801–1809, 1967.

[120] J. Unger, A. Kröner, and W. Marquardt. Structural analysis of differential-algebraic equation systems – theory and applications. *Computers & Chemical Engineering*, 19(8):867–882, 1995.

[121] R. S. Upadhye and E. A. Grens. An efficient algorithm for optimum decomposition of recycle systems. *AIChE Journal*, 18:533–539, 1972.

[122] Ravindra S. Upadhye and Edward A. Grens. Selection of decompositions for chemical process simulation. *AIChE Journal*, 21:136–143, 1975.

[123] G. V. Varma, K. H. Lau, and D. L. Ulrichson. A new tearing algorithm for process flowsheeting. *Computers & Chemical Engineering*, 17(4):355–360, 1993.

[124] James A. Vegeais and Mark A. Stadtherr. Parallel processing strategies for chemical process flowsheeting. *AIChE Journal*, 38(9):1399–1407, 1992.

[125] R.C. Vieira and E.C. Biscaia Jr. Direct methods for consistent initialization of DAE systems. *Computers & Chemical Engineering*, 25(910):1299–1311, 2001.

[126] M. Vlach. LU decomposition and forward-backward substitution of recursive bordered block diagonal matrices. *IEE Proceedings G (Electronic Circuits and Systems)*, 132:24–31, February 1985.

[127] Volker Waurich, Ines Gubsch, Christian Schubert, and Marcus Walther. Reshuffling: A symbolic pre-processing algorithm for improved robustness, performance and parallelization for the simulation of differential algebraic equations. In *Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, EOOLT '14, pages 3–10, New York, NY, USA, 2014. ACM.

[128] A. W. Westerberg and F. C. Edie. Computer-aided design, Part 1 Enhancing Convergence Properties by the Choice of Output Variable Assignments in the Solution of Sparse Equation Sets. *The Chemical Engineering Journal*, 2:9–16, 1971.

[129] A. W. Westerberg and F. C. Edie. Computer-Aided Design, Part 2 An approach to convergence and tearing in the solution of sparse equation sets. *Chem. Eng. J.*, 2(1):17–25, 1971.

[130] Arthur W. Westerberg and Thomas J. Berna. Decomposition of very large-scale Newton-Raphson based flowsheeting problems. *Computers & Chemical Engineering*, 2(1):61 – 63, 1978.

[131] R. J. Wilson. *Introduction to Graph Theory*. Pearson Education Limited, Harlow, England, 5th edition, 2010.

[132] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

[133] L. Zhou, Z. Han, and K. Yu. A new strategy of net decomposition in process simulation. *Computers & Chemical Engineering*, 12(6):581–588, 1988.