EFFICIENT ELIMINATION ORDERS FOR THE ELIMINATION PROBLEM IN DIAGNOSIS

ERIK FRISK

Abstract. A consistency relation is a constraint on the time evolution of known variables (and their time derivatives) that is fulfilled if the known variables are consistent with a model. Such relations are useful in diagnosis and can be derived using elimination theory. Unfortunately, even apparently small elimination problems proves impossible to compute on standard computers. An approach to lessen the computational burden is to divide the complete elimination problem into a set of smaller elimination problems. This is done by analysing the structure of the model equations using graph theoretical algorithms from the field of sparse factorization of symmetric matrices.

The algorithms are implemented in Mathematica and exemplified on a fluid-flow system where the original elimination problem does not terminate. Applying the proposed algorithms give an elimination strategy that terminates with a solution in just a few seconds.

1. Introduction

Consistency based diagnosis is performed by determining consistency and inconsistency between measured data and a model of the process under supervision. One way to evaluate consistency between model and measured data is by using precomputed *consistency relations*[1].

A consistency relation is a constraint on the time evolution of known variables (and their time derivatives) that is fulfilled if the known variables are consistent with the model. Based on a consistency relation it is possible to compute a *residual* which is a signal that ideally is 0 when the model is consistent with data and non-zero when data can not be explained by the model. Inconsistencies is, in the diagnosis application, ideally due to faults. By designing a carefully chosen set of such relations where each relation validates different submodels it is possible to derive a set of residuals each sensitive to different subsets of faults. In this way it is possible not only to detect faults, but also isolate faults. This well known approach to fault isolation is in the FDI-community known as *structured residuals*.

Deriving consistency relations can be done using a variable elimination approach. For linear systems variable elimination is easy and straightforward but for non-linear systems this is a much more difficult problem. Section 2 demonstrates a procedure to derive consistency relations by variable elimination. It also illustrates how the complexity of the elimination problems makes straightforward elimination on a standard personal computer and a computer algebra package unfeasible, even for apparently small models. Section 3 introduces the main ideas of how the elimination problem can be systematically divided into a set of smaller subproblems for which the elimination procedure is easier. This is done by a structural analysis of the model equations. Two graph theoretical algorithms for this is described in

Key words and phrases. diagnosis, elimination theory, consistency relations, structural analysis, elimination trees.

[1]Consistency relations are often also called Analytical Redundancy Relations (ARR).

Section 4 and the approach is exemplified on an example in Section 5. Finally, some conclusions in Section 6.

2. Deriving Consistency Relations by Variable Elimination

Before describing how consistency relations can be derived, a formal definition taken from [KN02] of consistency relations:

Definition 1 (Consistency relation)**.** *Let x and z be unknown and known variables respectively. Then, a scalar equation $c(z) = 0$ is a consistency relation for a set of equations $H(x, z)$ if and only if for all z it holds that*

$$\exists x.\, H(x, z) = 0 \Rightarrow c(z) = 0 \tag{1}$$

and there is no proper subset of H that has property (1)*.*

Relation $c(z) = 0$ is thus a test on consistency of equations $H(x, z)$ with available data. The task of the diagnosis system designer is then to find a set of such relations with a suitable influence from faults to facilitate also fault isolation. From a consistency relation $c(z) = 0$ a residual can be computed as

$$r = c(z)$$

which can be thresholded for a test of consistency. Note that for dynamic systems, the consistency relation normally includes time differentiated variables which are normally not known. The topic of how to estimate the derivatives to compute the residual is not pursued further here.

A requirement on the set of equations $H(x, z) = 0$ to be able to form a consistency relation is that it contains *redundancy*, i.e. the equations represents an over-constrained set of equations. Moreover, it is assumed that H is a minimal structurally over-constrained set of equations. This is formally defined as

Definition 2 (Minimal Structurally Singular, MSS)**.** *A finite set of equations H is structurally singular with respect to variables X if the number of equations is larger than the number of variables that appear in the equations, i.e. the set of equations is over-constrained with respect to X. The set H is a minimal structurally singular set if none of its proper subsets are structurally singular.*

This final assumption simplifies the algorithms proposed in Section 3. It is also important since if the set of equations is not an MSS, then it would be possible to derive a consistency relation by eliminating a subset of the variables in a subset of the equations, i.e. an unnecessary large equation set is used. Computing MSS sets before eliminating can greatly influence computation time, see e.g. [Fri01] for examples.

Therefore, a first step for the diagnosis system designer is to find a set of minimal over constrained subsystems with suitable fault influence structure, from which the consistency relations can be derived. Finding these subsets is not the topic of this report, see for example [KN02] for a structural approach to selecting the submodels to be evaluated. Rather, the main topic of this work is to, given a submodel, derive a consistency relation using elimination methods.

A variable elimination procedure to compute $c(z)$ is conceptually simple and is perhaps best shown by a small example.

Example 1. Consider a state-space description

$$\begin{aligned} \dot{x}_1 &= -x_1 x_2 + \gamma u \\ \dot{x}_2 &= x_3 \\ \dot{x}_3 &= -\gamma x_1 \\ y_1 &= x_1 \\ y_2 &= x_2 \end{aligned} \tag{2}$$

where γ models a fault we wish to decouple, i.e. a consistency relation independent of γ is wanted.

The following relations are immediate by differentiating both measurement equations and making obvious substitutions:

$$\begin{aligned} \dot{y}_1 + y_1 y_2 - \gamma u &= 0 \\ \ddot{y}_2 + \gamma y_1 &= 0 \end{aligned}$$

But none of these is a consistency relation since both relations include the unknown, possibly time-varying, variable γ. But γ can easily be eliminated by multiplying the first relation with y_1, the second with u, and adding the two. The resulting consistency relation is then

$$y_1 \dot{y}_1 + y_1^2 y_2 + u\ddot{y}_2 = 0$$

which is a nonlinear consistency relation that holds for all y_1, y_2, γ that satisfies model equations (2).

In the small example above, the elimination process was easy and could be performed by hand. For larger problems, support from computer algebraic tools is necessary. Therefore, the class of models considered from now on will be limited to systems where automatic elimination can be done by computer algebraic tools. This, more or less, limits the systems to polynomial systems and the elimination process comes down to computing a Gröbner basis for the elimination ideal. See [CLO96] for theoretical details on Gröbner bases and their properties.

However, even with this limitation in model description, the problem size need not be particularly large before the time and space-complexity of the Gröbner basis computations make elimination unfeasible on standard computers. It is not difficult to find small example models where a computer algebraic package on a standard PC[2] does not terminate. For example, a small model consisting of 4 randomized equations of second-order polynomials in 3 variables with symbolic coefficients, elimination proved too difficult and memory consuming. Memory (both physical and virtual) ran out before computations terminated.

However, a main point of this work is that physical systems are not randomized but often exhibits structure. Next section introduces a way to utilize this structure in the elimination problem to lessen the computational complexity.

3. Divide and Conquer by Structural Analysis

As noted in the last section, time and space complexity of computing Gröbner bases are quite large. This also means that partitioning the elimination problem into a set of smaller sub-problems would be an interesting approach. This section proposes a simple idea how the structure of the model equations can be analyzed to find a way to partition the elimination problem. As it shows, standard methods from the field of factorization of sparse matrices and parallel computing schemes come in handy for this. Descriptions of two standard ways to partition the elimination problem is provided in Section 4.

[2]The author used a Pentium II-200 MHz with 128 Mb of memory running Windows 2000.

If the system of equations were linear, dividing the elimination problem into a set of smaller elimination problems would always be a good idea. However, for general non-linear equations this might not always be the case and in particular problem instances, feeding a larger set of equations to the solver might produce a smaller solution in a shorter time than dividing the elimination into several steps. However, the authors experience is that for physically motivated models, a divide and conquer approach is fruitful and from now on, without any theoretical motivation, the basic assumption is that the analytical form of the model equations is such that it is a sound idea to divide the elimination problem into a set of smaller elimination problems.

To proceed and describe the main idea, we need an exact definition of what is meant by the structure of a model. Two representations, the incidence matrix and an incidence indicator function is used throughout the text.

Definition 3 (Incidence matrix/incidence indicator function)**.** *Let $\mathcal{X}$ be a set of variables and $\mathcal{C}$ a set of constraints. The incidence matrix for $\mathcal{C}$ with respect to $\mathcal{X}$ is a binary matrix S such that*

$$S_{i,j} = 1 \Leftrightarrow \textit{variable } i \textit{ appears in constraint } j$$

The incidence indicator function $\mathcal{S} : \mathcal{X} \times \mathcal{C} \to \mathcal{B} = \{\textit{True}, \textit{False}\}$ is defined as

$$\mathcal{S}(x_i, c_j) = \textit{True} \Leftrightarrow S_{i,j} = 1$$

The basic idea of the approach is perhaps most easily illustrated on a small example.

Example 2. Consider a set of four polynomial equations $e_1, \ldots, e_4$ in variables $x_1, \ldots, x_3$, and y. Variable y is known and the x_i:s are unknown. Also assume that the structure with respect to the unknown variables is according to the incidence matrix:

	x_1	x_2	x_3
e_1	1		1
e_2	1		
e_3		1	1
e_4		1	

The model consists of 4 equations and 3 unknown variables to be eliminated, i.e the equations form a (minimal) structurally singular set of equations. Based on the structure it is easy to see that variable x_1 can be eliminated using only equations e_1 and e_2 and x_2 using e_3 and e_4. This further means that variables x_1 and x_2 can be eliminated in parallel using only 2 equations each. This is due to the fact that they both appear in disjunct subsets of equations. The set $\{x_1, x_2\}$ are said to be an *independent* set of variables. The resulting equations after elimination are joined and used to eliminate the final variable x_3. This elimination strategy can be represented as the list $\{\{x_1, x_2\}, \{x_3\}\}$. Other elimination strategies are of course possible, however there exists no shorter strategy. In particular, elimination of x_3 before x_1 or x_2 gives a longer elimination strategy.

In the example above, the elimination order was easy to see directly in the incidence matrix. In the next section, two graph theoretic algorithms are presented that automatically, based on the model structure, computes good elimination strategies.

4. Graph-Theoretical Algorithms

This section will describe how to formulate the stated problem into finding a so called *elimination order*. For this problem, finding an elimination order with suitable properties, two standard approaches from the field of sparse factorization

is reviewed. Detailed descriptions of the basic concepts from sparse factorization can be found in e.g. [GL81].

In diagnosis, an often used graph theoretic description of the structural model is the *bi-partite* graph with equations and variables as vertices [SCD00]. Here, another graph description is used, suited for the problem posed here. A formal definition of this graph is given by:

Definition 4 (Graph induced by S)**.** *Let $\mathcal{S} : \mathcal{X} \times \mathcal{C} \to \mathcal{B}$ be an incidence indicator function. The vertices in the graph $G =< E, V >$ induced by $\mathcal{S}$ equal the set of variables $V = \mathcal{X}$ and the set of edges is defined by*

$$(x_i, x_j) \in E \Leftrightarrow \exists c \in \mathcal{C}.\ \mathcal{S}(x_i, c) \wedge \mathcal{S}(x_j, c)$$

The graph induced by incidence matrix S will be denoted $G(S)$, or only G if S is clear by context. The set of vertices of the graph is denoted $V(G)$ and the set of edges $E(G)$. This graph represent which variables that are directly connected through the equations and which variables that are only indirectly connected. Analysis of this graph is the basis for the algorithms to follow.

First it will be demonstrated how, using the graph introduced above, the problem of dividing the elimination problem can be stated as finding a "good" *elimination order*. It will also be demonstrated what is meant by a "good" elimination order. Since elimination orders is of central importance in this presentation, a proper definition is included.

Definition 5 (Elimination order of a graph)**.** *An elimination order is a complete order of the vertices in the graph.*

Variables ordered (or ranked) first are eliminated before variables that are ranked higher. Given an elimination order and the graph, an *elimination tree* can be computed in which it is easy to see which variables that can be eliminated in parallel. Thus, a "good" elimination order will be characterized by the resulting elimination tree. A formal definition of elimination trees and an example of how to compute the tree structure is given in Appendix A.

Consider for example the small set of equations in Example 2 using elimination orders $x_1 \prec x_2 \prec x_3$ and $x_3 \prec x_2 \prec x_1$. The corresponding elimination trees are shown in Figure 1. The first elimination tree is of height 1 and the second of height 2.

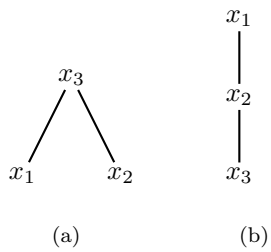


FIGURE 1. Elimination trees for Example 2 using elimination orders $x_1 \prec x_2 \prec x_3$ (a) and $x_3 \prec x_2 \prec x_1$ (b).

Also in the first tree, since vertices x_1 and x_2 are nodes in separate branches of the elimination tree they are independent and can be eliminated in parallel. It is evident that in the second tree, no such possibilities exists. A necessary and sufficient condition on when variables can be eliminated in parallel is given by Theorem 1.

Now we can state what is meant by a good elimination order. In sparse factorization, it is often desired that the elimination order minimizes what is called *fill*. Here the main objective is to get an order in which many variables can be eliminated independently, i.e. with as large parallelism as possible. Orders corresponding to such situations correspond to elimination trees of minimal (or low) height. The problem is therefore set to find an elimination order corresponding to minimal height elimination trees. Unfortunately, finding such elimination orders for general graphs has been proven NP-hard [Pot88] and we are forced to settle for heuristics finding low elimination trees. For special classes of graphs, e.g. when G is a tree or an interval graph, efficient algorithms have been designed that computes minimum height elimination tree orderings.

Here it is important to remember that it is assumed that the set of equations form an MSS, i.e. all equations are needed to eliminate the unknown variables. Also, it is structurally certain that it is not possible to choose a subset of equations to perform elimination of all variables. This further means that an elimination order must include *all* variables, i.e. a consistency relation can not be obtained eliminating only a subset of the variables. Also, under some regularity assumption on the analytic expressions it is also guaranteed that it is possible to eliminate all variables into a consistency relation.

Now, the following two sections will introduce two standard variable ordering algorithms. Here, a brief description is given. See for example [GL81; GL78; Geo73; LRT79] for detailed descriptions (although in a different problem setting) of their characteristics and theoretical properties.

4.1. **Nested Dissection Elimination Orders.** In order to get a low elimination tree we need to have as many branches as possible in the tree and also a well balanced tree. This basic observation is the basis for the variable ordering described in this section, nested dissection.

Vertices in different branches of an elimination tree are unrelated. Nested dissection is a top-down approach to achieve such a tree where the variable to eliminate *last* is found *first*. For this, an important property to characterize is when two variables are unrelated, i.e. can be eliminated in parallel. A necessary and sufficient condition is given by the following result:

Theorem 1. *Let G be a graph, α an elimination order, and $T_\alpha(G)$ the corresponding elimination tree and let x and y be two distinct nodes in $V(G)$ such that $x, y \neq root(T_\alpha)$. Let z be the lowest common proper ancestor of x and y in T_α, and let C be the set of all nodes on the path $z, \ldots, root(T_\alpha)$. Then x and y are unrelated in T_α if and only if C is a separator in G such that x and y are in different components of $G - C$.*

Proof. See [Liu90]. □

The result implies that if a separator C of graph G is found, the elimination tree will branch. This naturally leads to the *nested dissection* heuristic

Heuristic 1 (Nested Dissection Ordering)**.**

(1) Compute the graph G based on the structural model.
(2) Find a "good" separator of the graph. Order the variables in the separator last. If the graph is a clique, i.e. all vertices are connected to every other vertex, no separator exists and any elimination order gives the same elimination tree height (a single chain of all variables).
(3) Apply the heuristic recursively for all components in $G - C$.

A "good" separator is a small separator C such that $G-C$ has many components, i.e. many branches in the elimination tree, and where the components are roughly

the same size. The size requirement is imposed to produce a well balanced tree such that further branching further down in the elimination process is possible. Also note that

Lemma 1. *If S is the incidence matrix of an MSS E, then the graph G induced by S is connected.*

Proof. Assume that the graph G consist of more than one component, i.e. G is the union of a number of subgraphs $G_i =< V_i, E_i >$ with $V_i \cap V_j$ if $i \neq j$. The variable sets V_i partition the complete set of variables and, by definition, E is structurally singular. This means that there exist at least one component G_k such that $Equ_{V_k}(E)$ is also structurally singular. This is a contradiction since E is an MSS and no proper subset of E is structurally singular which ends the proof. □

Remark 1. The converse of this result is not true, a structurally singular set that is not minimal may or may not have a connected graph induced by its structure.

This means that when starting the heuristic we begin with a single connected component. The complete procedure is now illustrated on a small example.

Example 3. Consider a set of 9 equations in 8 unknown variables (known variables need not be included in the analysis) with the incidence matrix

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
c_1	1	0	1	0	0	0	0	0
c_2	1	0	0	0	1	0	0	0
c_3	0	0	1	0	0	1	0	0
c_4	0	0	0	0	1	1	0	0
c_5	0	1	0	0	1	0	0	0
c_6	0	0	0	1	0	1	0	0
c_7	0	1	0	1	0	0	1	0
c_8	0	0	0	0	0	0	1	1
c_9	0	0	0	0	0	0	1	1

Since we have more equations than unknowns, the set is over-constrained. It can also be proven that it is a minimally over-constrained set of equations, i.e. an MSS. This can be proved e.g. by removing an arbitrary equation and finding a perfect matching in the bi-partite graph. Figure 2 shows the bi-partite graph corresponding to S. Neither the incidence matrix nor the bi-partite graph gives, at first sight, little

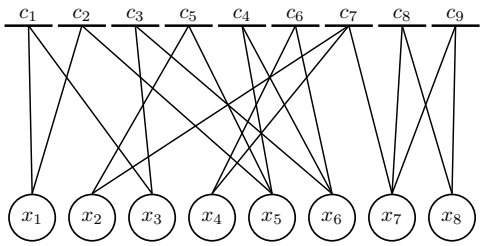


FIGURE 2. A bi-partite graph description of the MSS.

clue on which order to eliminate the variables. Figure 3 shows the graph induced by the structure S and here a possible elimination strategy becomes more clear. The procedure now is to find a good separator for this graph. There is only one separator consisting of a single vertex $C_1 = \{x_7\}$. However, this separates G into two ill-balanced components of sizes 1 and 6. A better choice is to use separator $C = \{x_2, x_6\}$. This separates G into two more well balanced components of sizes

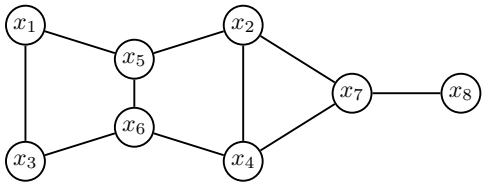


FIGURE 3. Graph induced by the structure S

3 and 3. For this reason are variables $x_2 \prec x_6$ ranked highest. The situation after the separator C has been removed is shown in Figure 4-a. The same procedure is then applied recursively on the two components and the separators for each component is $\{x_1\}$ and $\{x_7\}$ respectively. Finally, in the last step when also these separators have been removed we are left with four single vertices in Figure 4-b, for which separators of course does not exist. The final elimination order is then

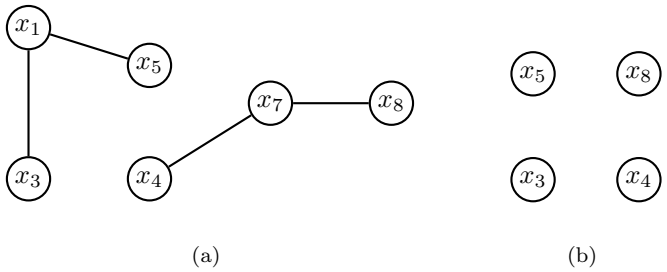


FIGURE 4. Graph after the first (a) and second (b) recursive call of the nested dissection heuristic where vertices $\{x_2, x_6\}$ and $\{x_1, x_7\}$ respectively has been removed.

$x_3 \prec x_5 \prec x_1 \prec x_4 \prec x_8 \prec x_7 \prec x_2 \prec x_6$ and the corresponding elimination tree is shown in Figure 5. From the elimination tree it is easy to deduce which

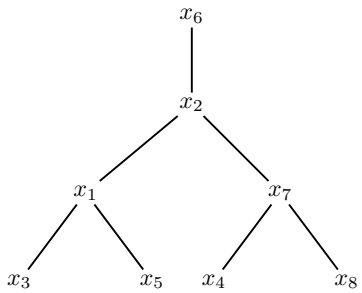


FIGURE 5. Elimination tree given by the nested dissection ordering.

variables that can be eliminated in parallel (or independently). All leafs in the elimination tree are independent according to Theorem 1. Therefore, in a first step can variables $\{x_3, x_4, x_5, x_8\}$ be eliminated independently. This is easily verified in the initial incidence matrix where the variables all occur in pairwise disjunct sets of equations. Remove these variables from the elimination tree and a new set of independent vertices are $\{x_1, x_7\}$. This is also possible to verify (but not as easy as in the first step) from the initial incidence matrix. In a third and final step, the last two variables x_2 and x_6 are eliminated.

4.2. **Maximal Independent Set Orders.** In the last section it was utilized that a low elimination tree have many branches. Another way of characterizing low trees is that a low tree have many leafs. Since leafs in an elimination tree corresponds to independent variables, a bottom-up approach can be used where at each step the maximum number of variables that can be eliminated in parallel are determined. These variables are ranked lowest. After these variables are eliminated and we have obtained a new model structure, the procedure is iterated.

The above heuristic will now be described using the graph induced by the model structure. Variables that can be eliminated in parallel are called independent variables and are formally defined as

Definition 6 (Independent Set of Vertices)**.** *A set of vertices I is said to be independent in the structural model S if and only if*

$$\forall v_1, v_2 \in I \Rightarrow (v_1, v_2) \notin E(G)$$

where G is the graph induced by S.

Note that this is equivalent to that the variables v_i all appear in pairwise disjunct sets of equations and therefore can be eliminated in parallel. To keep track of how the graph changes when variables are eliminated we utilize *elimination graphs*. See Definition 7 in Appendix A for a formal definition and a description. Now, the heuristic can be described as

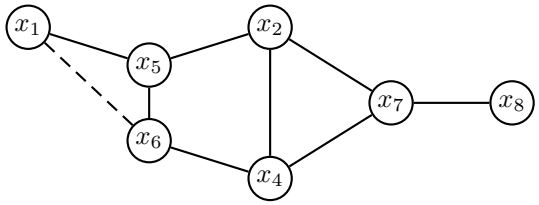
Heuristic 2 (Maximum Independent Sets Ordering)**.**

(1) Compute the graph G induced by the structure S
(2) Set $i = 1$
(3) If the variables of the G_i form a clique, i.e. all vertices are adjacent to all other vertices, order the variables in any order and exit.
(4) If the graph is not a clique, find the maximum independent set of variables C_i and order these (in any order) first.
(5) Compute the elimination graph where the variables in C_i have been eliminated.
(6) Iterate from 3 until all variables are eliminated
(7) The elimination order is then defined by $C_1 \prec C_2 \prec \dots$.

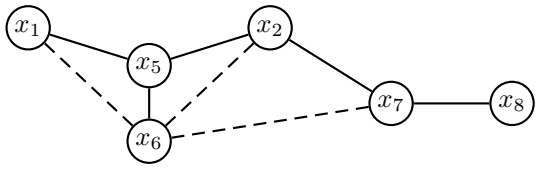
The maximum independent set heuristic is now demonstrated on the same model as was used in Example 3.

Example 4. The initial graph is of course the same as in Figure 3. The next step is to find the largest set of independent variables, i.e. variables that are not adjacent in the graph. The largest such set has 4 vertices, for example $\{x_3, x_4, x_5, x_8\}$. The next step is to compute the elimination graphs when these four variables are eliminated. Figure 6-a and 6-b show the result when x_3 and x_4 respectively have been eliminated. Introduced fill edges are dashed. Continuing in this fashion give that in a next step, the maximum set of independent variables are $\{x_1, x_7\}$. Finally, we are left with vertices x_2 and x_6 that form a clique. The final maximum independent set elimination order is thus $x_3 \prec x_4 \prec x_5 \prec x_8 \prec x_1 \prec x_7 \prec x_2 \prec x_6$. Note that this is not the same order as was found by the nested dissection procedure, however it can be verified that it corresponds to the same elimination tree, i.e. the tree in Figure 5.

4.3. **Final Comments.** The tools described in this section was originally designed to solve large scale systems of linear, symmetric, and positive definite equations. This makes an obvious difference to the non-linear diagnosis problem where we have, instead of maybe 50000 variables, perhaps at most 50 variables. This difference makes more elaborate search alternatives possible due to a drastically smaller



(a)



(b)

FIGURE 6. Elimination graphs when variables x_3 (a) and x_4 (b) are eliminated. Introduced fill edges are dashed.

problem size. Consider for example the nested dissection heuristic proposed in [GL78] and [GL81] based on the level structure of a pseudo peripheral vertex. This heuristic is very fast even for very large systems and give a decent separator. However, it is rather coarse and in Example 3, this ordering heuristic would give

$$ND : x_7 \prec x_8 \prec x_3 \prec x_5 \prec x_1 \prec x_6 \prec x_2 \prec x_4$$

which corresponds to an elimination tree of height 4 instead of height 3 found with a more thorough search heuristic. It is noteworthy to state that for problems this size (8 variables) a complete search[3] takes no more than approximately 40 seconds. A complete search seem to be feasable for up to $\approx 8 - 10$ variables, beyond that the problem size increase very quickly rendering exhaustive search impossible.

This also illustrates that, for a general graph, no nested dissection and no maximum independent set ordering will give a minimum height elimination tree. This fact is nicely illustrated in [Man91a; Man91b] where also an algorithm is proposed to try local reorderings of a given elimination order to reduce elimination tree height. The algorithm, minimal cutset reorderings, is guaranteed not to increase the height of the elimination tree. Such algorithms is here used in Section 5 to strengthen the assumption on minimum elimination tree height.

When an elimination order has been computed, the next step is to decide an elimination strategy. Assuming that the strategy is selected such that in each step, the largest set of independent variables are eliminated. This corresponds to the highest possible degree of parallelism in a given elimination order. The only thing left to decide is how to eliminate chains of variables, or equivalently cliques in the elimination graphs. For example, consider variables x_2 and x_6 in the elimination tree in Figure 5. These could, according to the structure, be eliminated in a single step or one at a time in any order. One straightforward heuristic could be to divide chains into a number of subchains of a maximum length and where the variables in

[3]Using the sparse matrices toolbox in Matlab.

each subchain is eliminated in a single step. The maximum length of a chain can be set according to available computing power.

5. Example: A Water Tank System

The variable ordering techniques described in the last section has been implemented in a computer algebraic language for Mathematica and the approach is in this section demonstrated on an example. Details on the exact implementation of the heuristics is not given here. Interested readers are referred to the code.

Consider the tank system depicted in Figure 7. The process is controlled by the

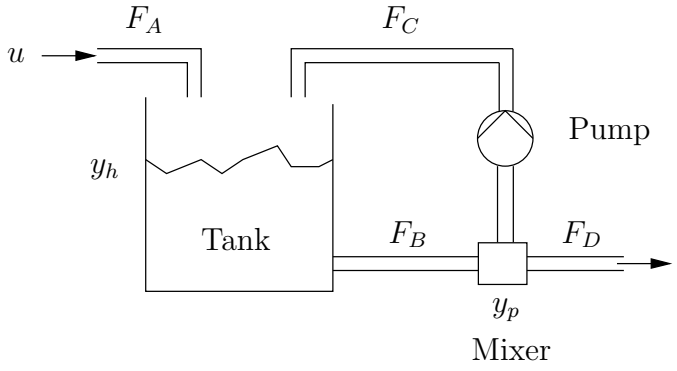


Figure 7. A small water tank system.

in-flow u and measured variables are the water level h in the tank and the pressure p_m in the mixer. A simple and normalized model can be stated as

$$
\begin{aligned}
e_1: &\quad F_A(t) = u(t) \\
e_2: &\quad 0 = h(t) + p_{atm} - F_B^2(t) - p_m(t) \\
e_3: &\quad 0 = p_m(t) - p_{atm} - F_C^2(t) + (1 - f_C(t)) \\
e_4: &\quad 0 = p_m(t) - p_{atm} - F_D^2(t) - f_D(t) \\
e_5: &\quad 0 = F_B(t) - F_C(t) - F_D(t) \\
e_6: &\quad \dot{h}(t) = F_A(t) + F_C(t) - F_B(t) \\
e_7: &\quad y_h(t) = h(t) + f_h(t) \\
e_8: &\quad y_p(t) = p_m(t)
\end{aligned}
$$

where F_A, F_B, F_C, and F_D are flows, p_{atm} is the atmospheric pressure, p_m the mixer pressure, and y_h the measured water level, and y_p the measured mixer pressure. There are three faults acting on the process, f_C, f_D, and f_h, each representing a fault in the mixer, a leakage, and a sensor fault respectively.

Assume that the objective is to derive a consistency relation for isolating the fault f_h from the other two faults, i.e. a consistency relation decoupling f_h is needed. A structural analysis as described in [KN02] give that

$$H = \{e_1, \dot{e}_2, e_3, \dot{e}_3, e_4, \dot{e}_4, e_5, \dot{e}_5, e_6, e_8, \dot{e}_8\}$$

form an MSS where fault f_h is decoupled. That H is structurally singular is direct since H includes 11 equations and 10 unknown variables, namely

$$\{p_m, \dot{F}_C, \dot{F}_D, F_A, F_B, \dot{h}, \dot{F}_B, \dot{p}_m, F_C, F_D\}$$

That f_h is decoupled in H is easily verified since f_h only appears in model equation e_7 which is not included in the MSS. The internal form of the consistency relation,

i.e. how the faults f_C and f_D enters the consistency relation, is also a desired result of the analysis. This means that variables f_C and f_D cannot be set to 0 in the model equation before eliminating. Doing so would reduce the (space-)complexity of the elimination problem.

Direct elimination of these 10 unknown variables proves impossible on the computer used by the author[4], the Gröbner basis computation never terminates. This motivates the use of techniques from the last section and the first step is to compute the graph induced by the model structure. This graph is shown in Figure 8. Computing both a nested dissection (ND) ordering and a maximal independent set

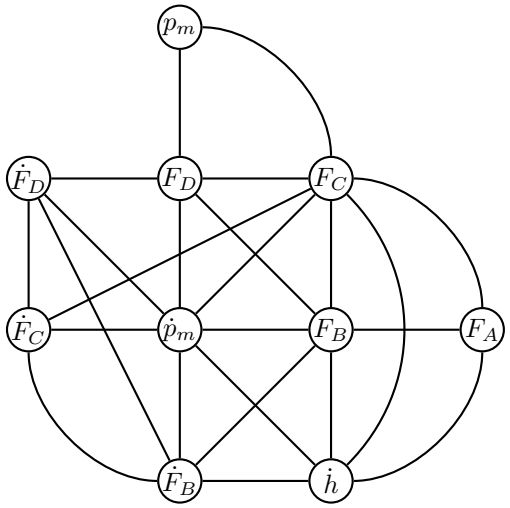


FIGURE 8. The graph induced by the structure of the water tank example model.

ordering (MI) for this model results in the orderings

$$ND : p_m \prec \dot{F}_C \prec \dot{F}_D \prec F_A \prec F_B \prec \dot{h} \prec \dot{F}_B \prec \dot{p}_m \prec F_C \prec F_D$$
$$MI : \dot{F}_D \prec \dot{h} \prec p_m \prec \dot{F}_C \prec F_B \prec \dot{p}_m \prec \dot{F}_B \prec F_D \prec F_C \prec F_A$$

and the corresponding elimination trees are shown in Figure 9. Both elimination trees are of equal height 6 and corresponds to elimination strategies

$$ND : \{\{F_A, \dot{F}_C, p_m\}, \{F_B, \dot{F}_D\}, \{\dot{h}\}, \{\dot{F}_B\}, \{\dot{p}_m\}, \{F_C\}, \{F_D\}\} \tag{3a}$$
$$MI : \{\{\dot{F}_D, \dot{h}, p_m\}, \{F_B, \dot{F}_C\}, \{\dot{p}_m\}, \{\dot{F}_B\}, \{F_D\}, \{F_C\}, \{F_A\}\} \tag{3b}$$

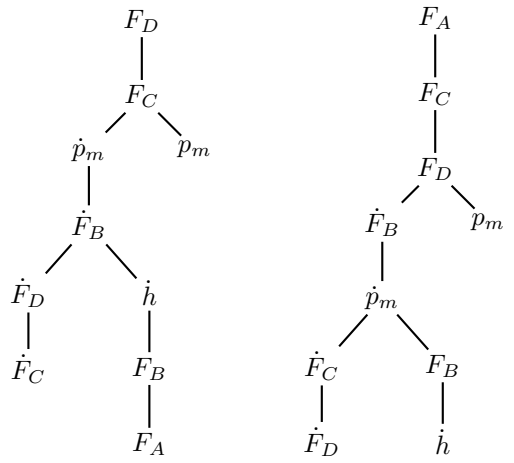
Thus, both orderings give elimination strategies of equal length and equal parallelism.

Performing the elimination according to any of the two strategies result (in 2-3 seconds) in a consistency relation $c(z) = 0$ (with p_{atm} set to 1)

$$\begin{aligned} c(z) = {} & y_p^6 + \dot{y}_p^4 + 2y_p\dot{y}_p^2(-1 + u^2 - 6u\dot{y}_p + 5\dot{y}_p^2) - 2y_p^5(2 + u^2 - 6u\dot{y}_p + 13\dot{y}_p^2) + \\ & + y_p^2((1+u^2)^2 - 12(u+u^3)\dot{y}_p + (30+44u^2)\dot{y}_p^2 - 48u\dot{y}_p^3 + 15\dot{y}_p^4) - \\ & - 2y_p^3(2 + 3u^2 + u^4 - 6u(3+2u^2)\dot{y}_p + (40+46u^2)\dot{y}_p^2 - 60u\dot{y}_p^3 + 25\dot{y}_p^4) + \\ & + y_p^4(6 + 6u^2 + u^4 - 12u(3+u^2)\dot{y}_p + (78+46u^2)\dot{y}_p^2 - 60u\dot{y}_p^3 + 25\dot{y}_p^4) \end{aligned}$$

which is a relation in the known variables y_p, $\dot{y}_p$, and u only.

[4]Pentium II-200 MHz with 128 Mb of memory running Windows 2000.



(a) $T_{ND}(G)$ (b) $T_{MI}(G)$

FIGURE 9. Elimination tree for the nested dissection ordering (a) and the maximum independent set ordering (b).

As was discussed in Section 4.3, neither nested dissection or maximum independent sets are guaranteed to deliver orders corresponding to minimum height elimination trees. For example, the ordering

$$\dot{p}_m \prec p_m \prec F_A \prec \dot{F}_D \prec \dot{F}_C \prec \dot{F}_B \prec \dot{h} \prec F_D \prec F_C \prec F_B$$

is also a maximum independent set ordering. This order however, results in an elimination tree of height 7. This difference is because in the first step when computing the MI-order, several independent set of vertices of equal cardinality are found. In this greedy algorithm they are considered equal, but in a next step they turn out to have different properties.

A way to decrease the height of an elimination tree is e.g. by using local reorderings of the elimination orders. But in this case, for example in the nested dissection ordering, all chains are cliques which means that none of the reordering algorithms in [Man91b] will reduce the height of the elimination tree. A similar result holds for the tree corresponding to the maximum independent set ordering. Therefore it is likely, but not proven, that a minimum height elimination tree for this problem has height 6 and that no shorter elimination strategy exists than (3).

The conclusion of this example is that, in this case, derivation of a consistency relation by computing a Gröbner basis involving all equations in the MSS is not possible. Instead, a divide and conquer approach was used and the computation time was reduced to a few seconds.

6. CONCLUSIONS

Elimination of unknown variables to derive a consistency relation for fault diagnosis has been considered. It is shown that, even apparently small elimination problems proves impossible to compute on a standard PC. A basic assumption of this work is that it is sound to divide the elimination problem into a set of smaller elimination problems. For this, several graph-theoretic heuristics from the field of sparse matrix factorization is presented. In sparse factorization, the problem size is huge, including possibly tens of thousands of variables, while in the diagnosis

application typically no more than 10-20 variables appear in the problem. Consequences of this change of problem size if discussed and how this can be utilized for diagnosis.

The algorithms has been implemented in Mathematica and tested on a model of a fluid flow system. For the problem posed in the example the elimination problem boils down to eliminating 10 variables using 11 non-linear equations. Feeding the complete problem into Mathematica does not terminate, but when using the proposed algorithms the elimination problem terminates with a consistency relation in just a few seconds.

Appendix A. Basic Notions From Graph Theory and Sparse Factorization

This appendix includes a brief description and formal definitions of some concepts used in the text. Most definitions are taken from [GL81]. Elimination trees are used heavily in the text and is of central importance, but before elimination trees can be defined, we need some additional tools.

Definition 7 (Elimination Graphs)**.** *Given an initial graph G_0 and a sequence of variables $x_1 \prec \cdots \prec x_n$, a sequence of elimination graphs can be obtained according to the recursion:*

1. *From graph G_i, remove vertex x_i and all its incidient edges.*
2. *Add edges so that all vertices in Adj (x_i) are pairwise adjecent.*

The resulting graph G_j is the j:th elimination graph.

The interpretation of this is straightforward, when eliminating the variable x_i all variables adjacent to x_i will be connected after the elimination. This is of course under the assumption that no "accidental" eliminations occur, i.e. that when eliminating x_i also variable $x_j \succ x_i$ is eliminated. Edges included in the elimination graphs, not included in the original graph is called *fill* and represent variable connections that occur during the elimination process that was not present in the original model.

Definition 8 (Filled Graph)**.** *The filled graph of graph $G =< X, E >$ with respect to elimination order α is denoted $G^\alpha =< X^\alpha, E^\alpha >$. The set of vertices of the filled graph equal the set of vertices in the original graph. The edges equal the edges in the original graph and the fill edges introduced during the elimination process.*

The filled graph represent relations between variables that exists in the original graph and also relations that are introduced during the elimination process. Figure 10 show an example of a filled graph. Now, with the definition of a filled graph,

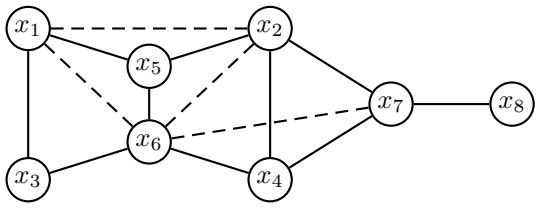


Figure 10. Filled graph of the graph in Figure 3 with respect to the elimination order $x_3 \prec x_5 \prec x_1 \prec x_4 \prec x_8 \prec x_7 \prec x_2 \prec x_6$. Introduced fill edges are dashed.

a formal definition of an elimination tree is straightforward:

Definition 9 (Elimination tree)**.** *Node x_j is the parent of node x_i if and only if $x_i \prec x_j$ and x_j is the lowest ranked variable among the higher ranked neighbours of x_i in the filled graph G^α.*

Consider again the graph in Figure 3 and the nested dissection elimination order $x_3 \prec x_5 \prec x_1 \prec x_4 \prec x_8 \prec x_7 \prec x_2 \prec x_6$. From the definition of an elimination tree and Figure 10 it is now straightforward to compute the parent-function $p(x)$

$$p(x_1) = x_2 \qquad p(x_2) = x_6 \qquad p(x_3) = x_1 \qquad p(x_4) = x_7$$
$$p(x_5) = x_1 \qquad p(x_6) = - \qquad p(x_7) = x_2 \qquad p(x_8) = x_7$$

which corresponds to the elimination tree in Figure 5.

Definition 10 (Tree height)**.** *The height of a rooted tree equal the length of the longest path from the root to a leaf.*

Definition 11 (Clique)**.** *A set of vertices $\mathcal{V}$ form a* clique *in graph $G =< X, E >$ if and only if*

$$v_i, v_j \in \mathcal{V},\ i \neq j \Rightarrow (v_i, v_j) \in E$$

Definition 12 (Chain)**.** *A* chain *$v_1, \ldots, v_n$ in a tree T is a path such that each node, with the exception if the last, has exactly one child.*

References

[CLO96] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra*, second ed., Springer Verlag, 1996.

[Fri01] E. Frisk, *Residual generation for fault diagnosis*, Ph.D. thesis, Linköping University, Sweden, 2001.

[Geo73] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis **10** (1973), no. 2, 345–363.

[GL78] A. George and J.W. Liu, *An automated nested dissection algorithm for irregular finite element problems*, SIAM Journal on Numerical Analysis **15** (1978), no. 5, 1053–1069.

[GL81] ———, *Computer solution of large sparse positive definite systems*, Series in Computational Mathematics, Prentice-Hall, 1981.

[KN02] M. Krysander and M. Nyberg, *Structural analysis for fault diagnosis of DAE systems utilizing graph theory and MSS sets*, Tech. Report LiTH-R-2410, Department of Electrical Engineering, Linköping University, SE-581 83 Linköping, Sweden, 2002.

[Liu90] L.W. Liu, *The role of elimination trees is sparse factorization*, SIAM Journal of Matrix Analysis and Applications (1990), no. 11, 134–172.

[LRT79] R.J. Lipton, D.J. Rose, and R.E. Tarjan, *Generalized nested dissection*, SIAM Journal on Numerical Analysis **16** (1979), no. 2, 346–358.

[Man91a] F. Manne, *Reducing the height of an elimination tree through local reorderings*, Fourth SIAM Triennial Conference on Applied Linear Algebra, 1991.

[Man91b] ———, *Reducing the height of an elimination tree through local reorderings*, Tech. Report CS-91-51, Department of informatics, University of Bergen, Norway, 1991.

[Pot88] A. Pothen, *The complexity of optimal elimination trees*, Tech. Report CS-88-13, Computer Science, Pennsylvania State University, 1988.

[SCD00] M. Staroswiecki, M. Cassar, and J.PH. Declerck, *A structural framework for the design of FDI in large scale industrial plants*, Issues of fault diagnosis for Dynamic Systems ed. R. Patton, P. Frank, and R. Clark, Springer Verlag, 2000.

Department of Electrical Engineering, Linköping University, SE-584 31 Linköping, Sweden
E-mail address: frisk@isy.liu.se